# CentOS7.2 – Docker 容器虚拟化

## Docker认知了解:

**Docker理念:** 一次封装,到处运行

解决了运行环境和配置问题软件容器，方便做持续集成并有助于整体发布的容器虚拟化技术

**虚拟机的缺点:** 1.资源占用多2.冗余步骤多3.启动慢

**比较 Docker 和 传统虚拟化 方式的不同之处:**

\*传统虚拟机技术是虚拟出一套硬件后,在其上运行一个完整操作系统,在该系统上再运行所需应用进程

\*而容器内的应用进程直接运行于宿主的内核,容器内没有自己的内核,而且也没有进行硬件虚拟,因此容器要比传统虚拟机更为轻便

\*每个容器之间互相隔离,每个容器有自己的文件系统,容器之间进程不会相互影响,能区分计算资源

**使用Docker容器虚拟化技术的好处:**

1. 更快速的应用交付和部署
2. 更便捷的升级和扩缩容
3. 更简单的系统运维
4. 更高效的计算资源利用

**Docker网址:**

|  |
| --- |
| docker官方网站: <http://www.docker.com>  docker中文网站: <https://www.docker-cn.com/>  docker Hub仓库: https://hub.docker.com/ |

**Docker三要素:** 仓库(repository) 镜像(image) 容器(container)

仓库(repository)是**集中存放镜像文件**的场所

仓库(repository)和仓库注册服务器(registry)是有区别的.仓库注册服务器上往往存放着多个仓库,每个仓库中又包含了多个镜像，每个镜像有不同的标签(tag)

仓库分为公开仓库(public)和私有仓库(private)两种形式

最大的**公开仓库是:Docker Hub(https://hub.docker.com/)**,存放了数量庞大的镜像供用户下载,国内的公开仓库包括阿里云 、网易云 等

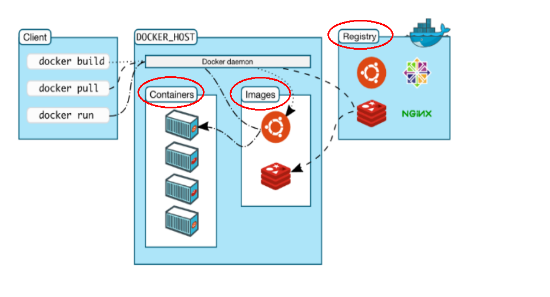
Docker镜像(image)就是一个**只读**的模板,镜像可以用来创建Docker容器,**一个镜像可以创建很多容器**

Docker容器(container)就是独立运行的一个或一组应用,**容器是用镜像创建的运行实例**

容器可以被启动、开始、停止、删除,每个容器都是相互隔离的,保证安全的平台

**可以把容器看做是一个简易版的 Linux 环境**(包括root用户权限、进程空间、用户空间和网络空间等)和运行在其中的应用程序

**Docker架构图:**



**容器与镜像的关系类似与面向对象编程中的对象与类:**

|  |  |
| --- | --- |
| **Docker** | **面向对象** |
| 容器 | 对象 |
| 镜像 | 类 |

**正确的理解仓储/镜像/容器这几个概念:**

Docker本身是一个容器运行载体或称之为管理引擎.我们把应用程序和配置依赖打包好形成一个可交付的运行环境,这个打包好的运行环境就是image镜像文件,只有通过这个镜像文件才能生成Docker容器,image文件可以看作是容器的模板,Docker根据image文件生成容器的实例,同一个image文件,可以生成多个同时运行的容器实例

\* 镜像文件生成的容器实例,本身也是一个文件,称为镜像文件

\* 一个容器运行一种服务,当我们需要的时候,就可以通过docker客户端创建一个对应的运行实例,也就是我们的容器

\* 至于仓储,就是放了一堆镜像的地方,我们可以把镜像发布到仓储中,需要的时候从仓储中拉下来就可以了

## Docker安装测试:

### 安装docker:

参考手册: <https://docs.docker-cn.com/engine/installation/linux/docker-ce/centos/#prerequisites>

安装准备:

yum -y install gcc

yum -y install gcc-c++

yum install -y yum-utils device-mapper-persistent-data lvm2

# yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo

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

yum makecache fast

安装命令: yum -y install docker-ce

### 启动docker:

启动: systemctl start docker

### 测试是否安装成功:

测试: docker version

### 配置镜像加速器:

创建目录: mkdir -p /etc/docker

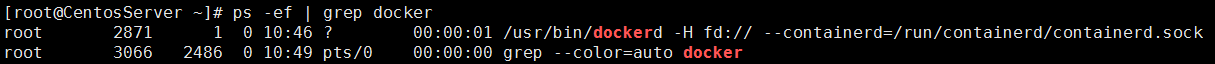
创建文件: vi /etc/docker/daemon.json 并添加如下内容

|  |
| --- |
| # 网易云  {  "registry-mirrors": ["http://hub-mirror.c.163.com"]   }  # 阿里云  {  "registry-mirrors": ["https://1qlfqh72.mirror.aliyuncs.com "]  } |

重新加载: systemctl daemon-reload

重新启动: systemctl restart docker

检测镜像加速是否生效: ps -ef | grep docker



### 重启docker:

重启: systemctl restart docker 或 service docker restart

### 卸载docker:

停止: systemctl stop docker

卸载: yum -y remove docker-ce

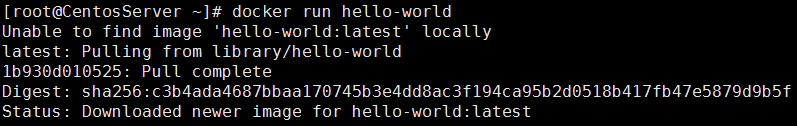
删除: rm -rf /var/lib/docker

### 运行状态docker:

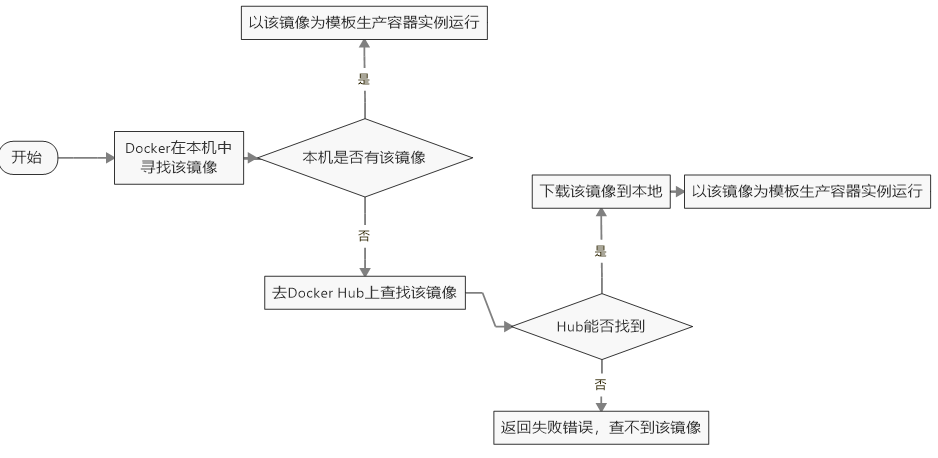
状态: systemctl status docker

### 测试运行hello-world:

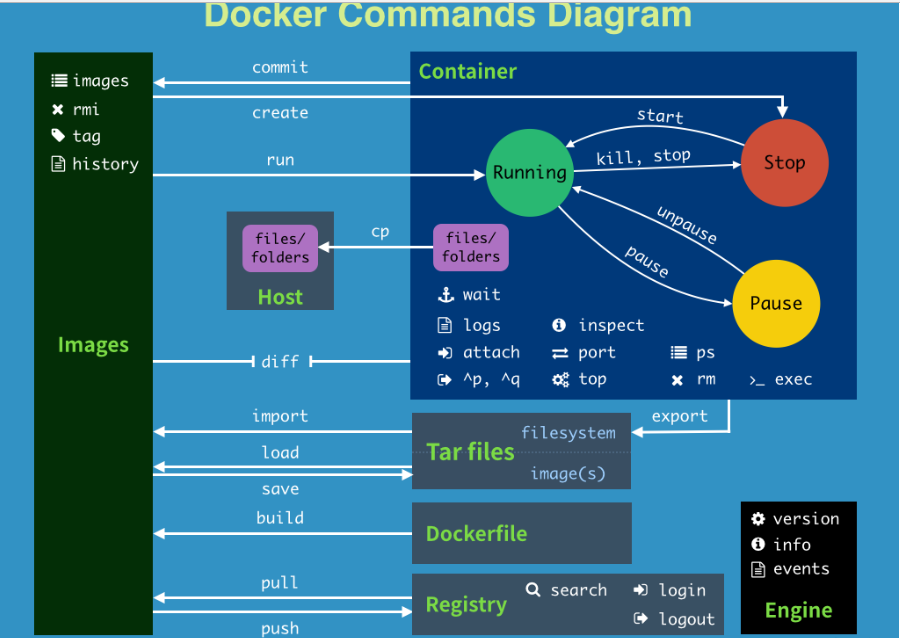
命令: docker run hello-world



**# run都干了什么操作:**



## Docker常用命令:



### 帮助命令:

Docker验证: docker version

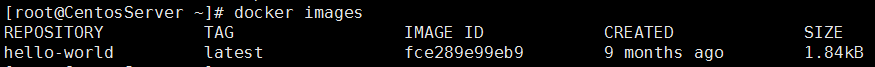
Docker信息: docker info

Docker辅助命令查询: docker –help

## Docker镜像命令:

### 列出本机镜像: docker images

命令: docker images



|  |  |
| --- | --- |
| REPOSITORY | 表示镜像的仓库源 |
| TAG | 镜像的标签 |
| IMAGE ID | 镜像ID |
| CREATED | 镜像创建时间 |
| SIZE | 镜像大小 |

同一仓库源可以有多个TAG,代表这个仓库源的不同个版本,我们使用 REPOSITORY:TAG 来定义不同的镜像

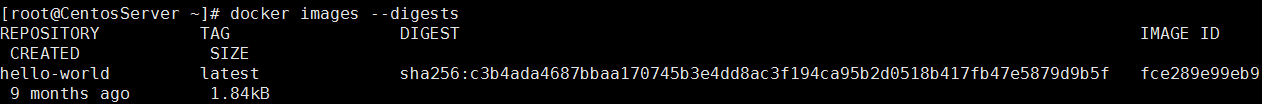
如果不指定一个镜像的版本标签,Docker将默认使用 Xxx:latest 镜像(最新版本)

参数信息说明:

|  |  |
| --- | --- |
| docker images -a | -a: 列出本地所有的镜像(含中间映像层) |
| docker images -q | -q: 只显示镜像ID |
| docker images --digests | --digests: 显示镜像的摘要信息 |
| docker images –no-trunc | --no-trunc: 显示完整的镜像信息 |

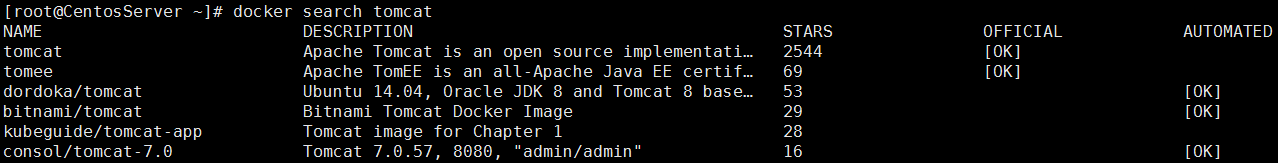






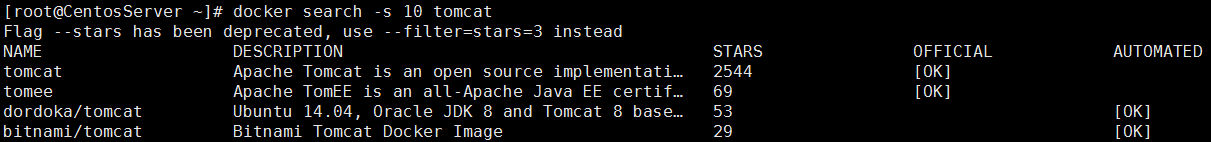
### 查找某个镜像: docker search xxx镜像名称

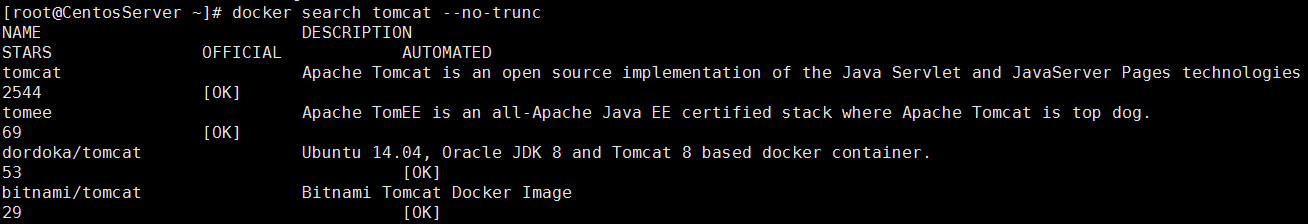
命令: docker search tomcat

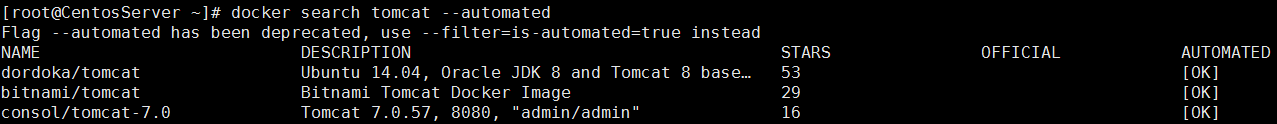


参数说明:

|  |  |
| --- | --- |
| docker search –s 10 tomcat | -s: 列出收藏数不小于指定值的镜像 |
| docker search tomcat --no-trunc | --no-trunc: 显示完整的镜像描述 |
| docker search tomcat --automated | --automated: 只列出 automated build 类型的镜像 |



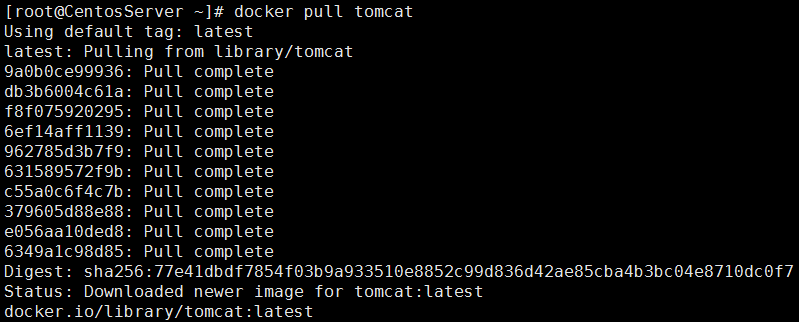




### 下载某个镜像: docker pull xxx镜像名称[:TAG]

命令: docker pull tomcat 等价于: docker pull tomcat:latest(最新版)



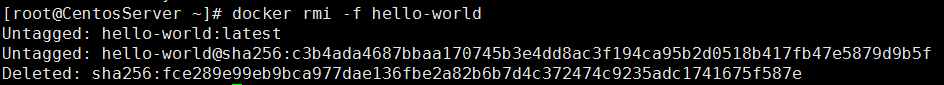


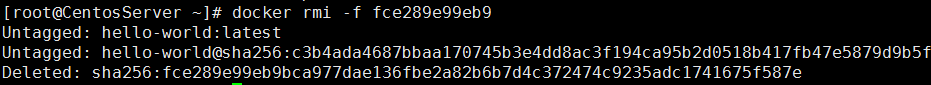
### 删除镜像: docker rmi -f xxx镜像名称[:TAG]/镜像ID

命令: docker rmi -f xxx镜像名称[:TAG] /镜像ID

注: docker rmi -f tomcat默认是删除latest标签的tomcat: 即 docker rmi -f tomcat:latest

删除单个: docker rmi -f 镜像名称[:TAG]/镜像ID

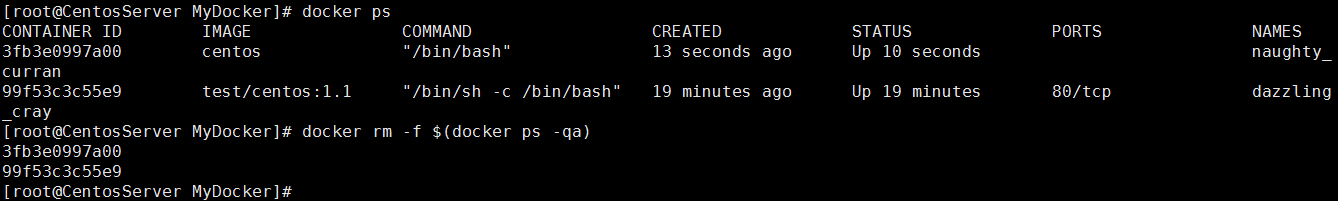




删除多个: docker rmi -f 镜像名称1[:TAG]/镜像ID 镜像名称2[:TAG]/镜像ID



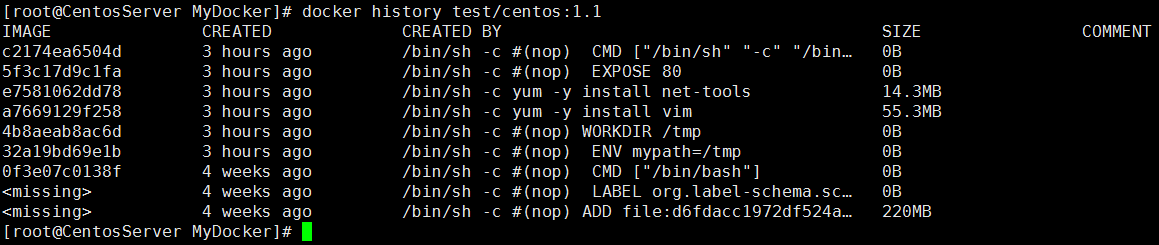
删除全部: docker rmi -f $(docker images -qa)



### 镜像变更历史:

自定义镜像: docker history 自定义镜像名称/自定义镜像ID

如: docker history test/centos:1.1



## Docker容器命令: ★

命令: docker run [opticons] 镜像名称/镜像ID

参数说明: options

|  |  |  |  |
| --- | --- | --- | --- |
| -i(交互) | | 以交互模式运行容器,通常与 -t 同时使用 | |
| -t(终端) | | 为容器重新分配一个伪输入终端,通常与 -i 同时使用 | |
| --name="容器新名字" | | 为容器指定一个名称 | |
| -d | | 后台运行容器，并返回容器ID,也即启动守护式容器 | |
| -P(大写) | 随机端口映射(随机分配端口) | | 如: docker run -it -P tomcat |
| -p(小写) | 指定主机端口:镜像的容器端口 | | 如: docker run -it -p 8888:8080 tomcat |

### 创建容器并启动容器:

**创建并启动容器: 以终端交互式方式创建容器并启动容器**

命令: docker run -it centos 或 docker run -it 0f3e07c0138f



**创建并启动容器: 以终端交互式方式创建容器并启动容器,且给容器取个别名**

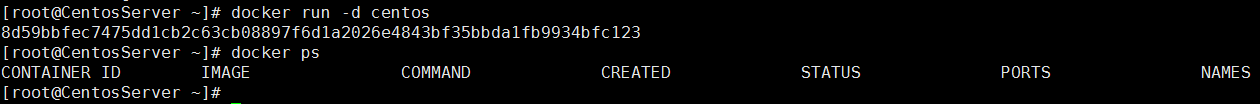
命令: docker run -it --name= "MyCentOS" centos



**创建并启动容器: 守护进程方式创建容器并启动容器**

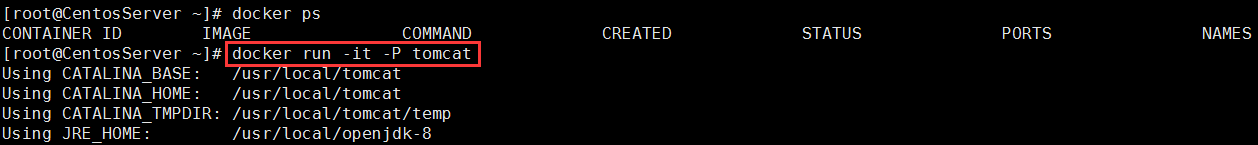
命令: docker run -d centos

命令: docker run -d -p 8888:8080 tomcat

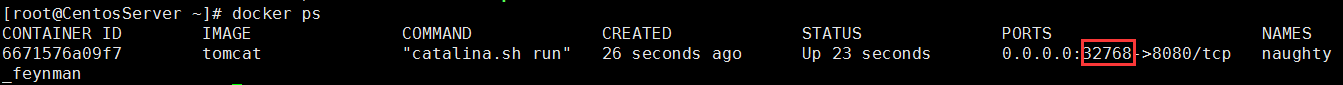


**创建并启动容器:(大p) 随机主机端口(镜像的容器使用默认端口)**

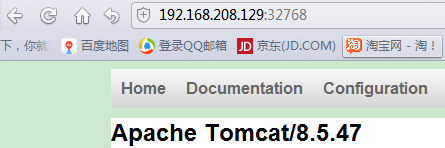
命令: docker run -it -P tomcat



重新再次打开一个终端查询: docker ps

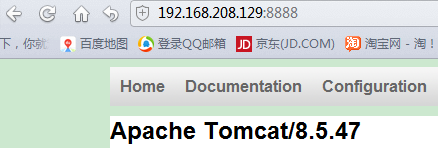


根据查询到端口32768访问tomcat: <http://IP地址:32768>



**创建并启动容器:(小p) 指定主机端口及镜像的容器端口,以便外部访问**

命令: docker run -it -p 8888:8080 tomcat

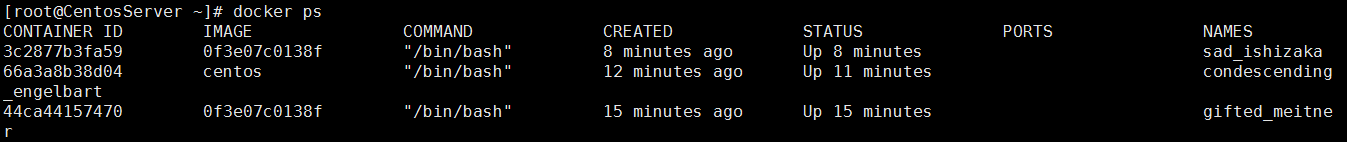


### 列出当前运行的容器:

命令: docker ps [opticons]

参数说明: options

|  |  |
| --- | --- |
| docker ps -a | -a: 列出当前所有正在运行的容器 + 历史上运行过的 |
| docker ps -l | -l: 显示最近创建的容器(即上一个容器) |
| docker ps -n 数字 | -n: 显示最近n个创建的容器 |
| docker ps -q | -q: 静默模式,只显示容器ID编号 |
| docker ps –no-trunc | --no-trunc: 不截断输出 |



### 退出容器:

容器停止退出: exit

容器不停止退出: ctrl + p + q 可以重新再进入容器

### 启动容器:

命令: docker start 容器ID/容器名称

如: docker start 66a3a8b38d04



### 重启容器:

命令: docker restart 容器ID/容器名称

如: docker restart 66a3a8b38d04



### 停止容器:

命令: docker stop 容器ID/容器名称

如: docker stop 66a3a8b38d04



### 强制停止运行容器:

命令: docker kill 容器ID/容器名称



### 删除已停止的容器:

删除一个已停止的容器: docker rm 7371c90b7c8d

删除多个已停止的容器: docker rm 7371c90b7c8d 6571c90b7c8d

删除所有已停止的容器: docker rm -f $(docker ps -aq) 或 docker ps -aq | xargs docker rm

### ★容器重要★:

#### 启动守护式容器:

命令: docker run -d 镜像名称/镜像ID

命令: docker run -d -p 8888:8080 tomcat



#### 查看容器的日志:

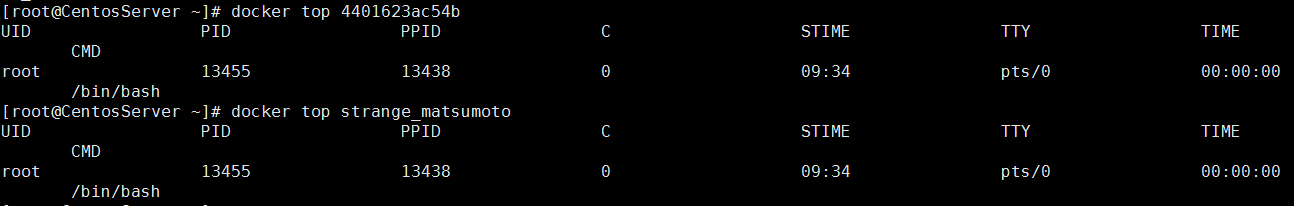
命令: docker logs -t -f -till num 容器ID/容器名称

参数说明:

|  |  |
| --- | --- |
| -t | 是加入时间戳 |
| -f | 跟随最新的日志打印 |
| --tail num | 数字 显示最后多少条 |

#### 查看容器内运行的进程:

命令: docker top容器ID/容器名称



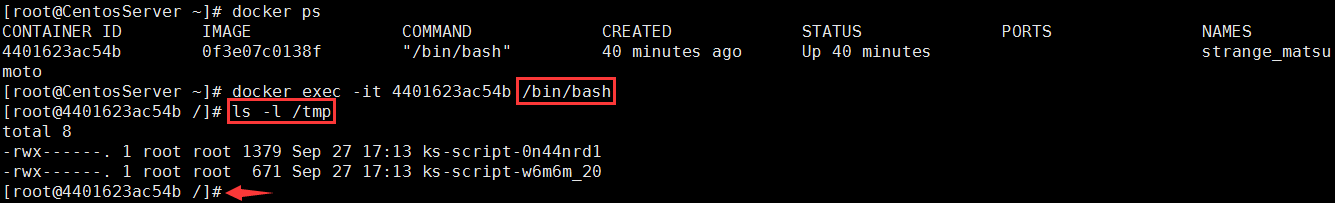
#### 查看容器内的内部细节:

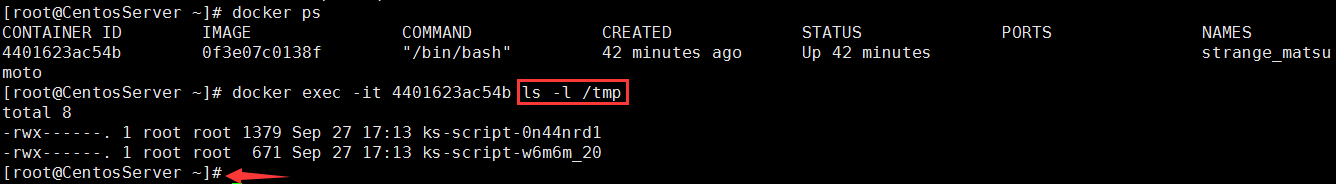
命令: docker inspect容器ID/容器名称

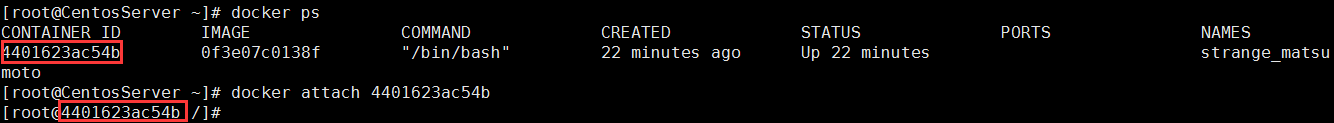
#### 进入正在运行的容器并以命令行交互:

命令:

|  |  |  |
| --- | --- | --- |
| docker exec -it 容器ID /bin/bash | | 进入容器中打开新的终端,并且可以启动新的进程 |
| docker exec -it 容器ID ls -l /tmp | | 进入容器并执行命令,显示结果(不进入容器的终端) |
| docker attach 容器ID | attach: 重新进入已启动容器的命令行终端,不会启动新的进程 | |

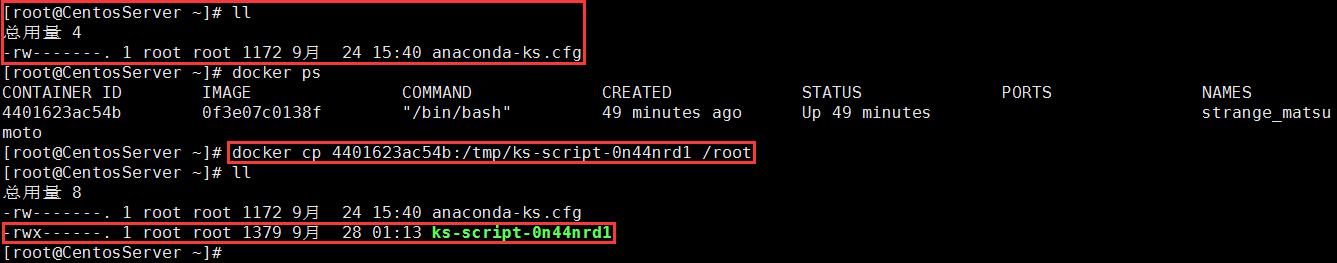






#### 从容器内拷贝文件到主机上:

命令: docker cp 容器ID/容器名称:容器内路径 目的主机路径



## Docker镜像操作: commit

镜像是一种轻量级,可执行的独立软件包,用来打包软件运行环境和基于运行环境开发的软件,它包含运行某个软件所需的所有内容,包括代码,运行时,库,环境变量和配置文件

UnionFS(联合文件系统):Unio文件系统支持对文件系统的修改作为一次提交来一层层的叠加

Union文件系统是 Docker 镜像的基础;镜像可以通过分层来进行继承,基于基础镜像(没有父镜像),可以制作各种具体的应用镜像

**docker镜像采用分层结构的好处:**

最大的一个好处就是 - 共享资源

比如:有多个镜像都从相同的 base 镜像构建而来,那么宿主机只需在磁盘上保存一份base镜像,同时内存中也只需加载一份 base 镜像,就可以为所有容器服务了;而且镜像的每一层都可以被共享

**示例如下: 提交容器副本使之成为一个新的镜像:**

**命令: docker commit -m=”提交的描述信息” -a=“作者” 容器ID/容器名字 新镜像名称:[标签TAG]**

### 1. 创建一个容器并启动容器,可以对容器的内容做修改:

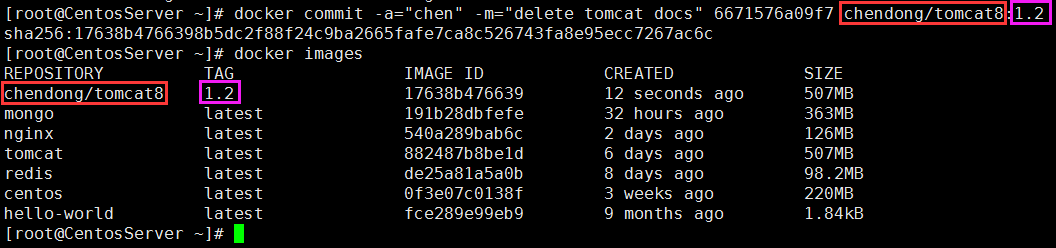
使用大P参数随机主机的端口(镜像的容器使用默认端口),如: docker run -it -P tomcat

使用小p参数指定主机的端口以及镜像的容器端口,如: docker run -it -p 8888:8080 tomcat

…修改(略)…

### 2. 把修改过后的容器副本作为一个新的镜像文件:

命令: docker commit -a="chen" -m="delete tomcat docs" 6671576a09f7 chendong/tomcat8:1.2

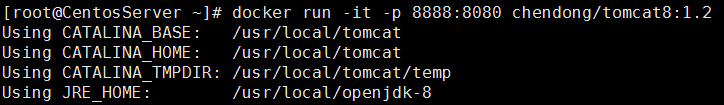


### 3.使用自定义镜像创建容器并启动:

注: TAG标签一定要正确

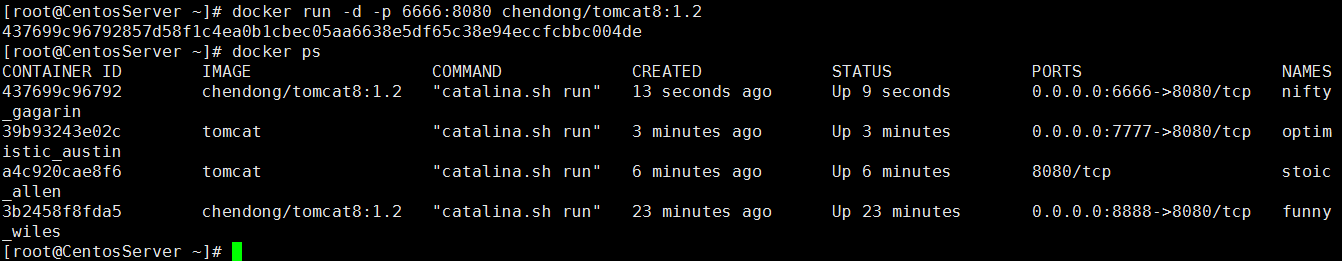
**# 以交互式(前台)的方式启动容器**

命令: docker run -it -p 8888:8080 chendong/tomcat8:1.2



**# 以守护式(后台)的方式启动容器**

命令: docker run -d -p 6666:8080 chendong/tomcat8:1.2



## Docker容器数据卷: ★

### 容器数据卷的作用:

1.容器的持久化 2. 容器间继承 + 共享数据

### 容器数据卷的特点:

1:数据卷可在容器之间共享或重用数据

2:数据卷中的更改可以直接生效

3:数据卷中的更改不会包含在镜像的更新中

4:数据卷的生命周期一直持续到没有容器使用它为止

### 容器数据卷的添加方式: (两种方式)

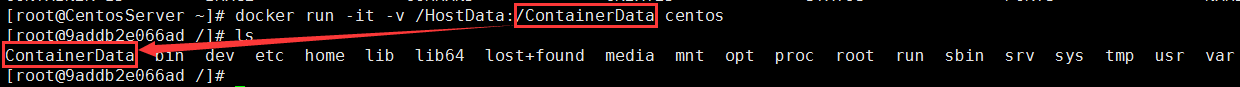
#### 使用命令添加: -v

**# 不带权限: 可读可写**

**添加命令:** docker run -it -v /宿主机绝对路径目录:/容器内的绝对路径目录 镜像名称/镜像ID

如: docker run -it -v /HostData:/ContainerData centos

容器Container:

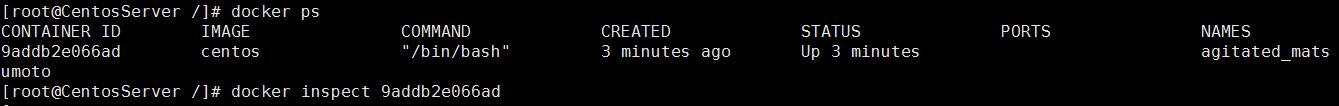


宿主机Host:



**是否挂载成功:**

命令: docker inspect 容器ID



**# 带权限: 只读不可写(针对容器)**

**添加命令:** docker run -it -v /宿主机绝对路径目录:/容器内的绝对路径目录:ro 镜像名称/镜像ID

如: docker run -it -v /HostData:/ContainerData:ro centos

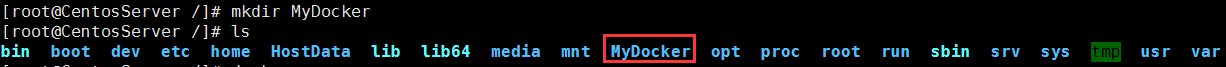
注: Docker挂载主机目录Docker访问出现cannot open directory .: Permission denied???

解决办法:在挂载目录后多加一个--privileged=true参数即可

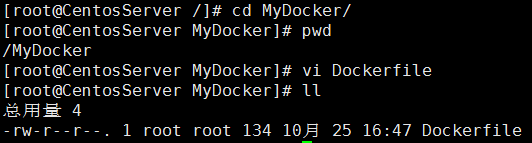
如: docker run -it -v /HostData:/ContainerData --privileged=true centos

#### 使用文件添加: Dockerfile

1. 在宿主机的根目录下创建MyDocker目录: mkdir MyDocker



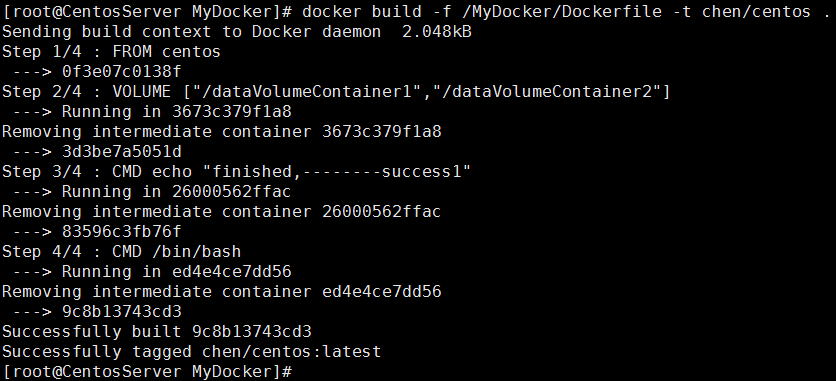
1. 在Dockerfile中使用volume指令给镜像添加一个或多个数据卷(粗略使用)



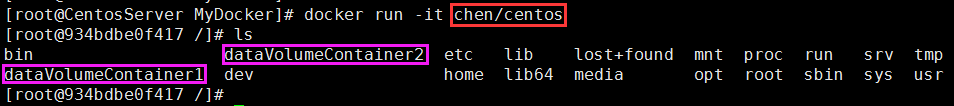
|  |
| --- |
| # volume test  FROM centos  VOLUME ["/dataVolumeContainer1","/dataVolumeContainer2"]  CMD echo "finished,--------success1"  CMD /bin/bash |

1. 根据Dockerfile文件使用build构建一个新的镜像

命令: docker build -f /MyDocker/Dockerfile -t chen/centos .



1. 以构建的新镜像创建容器并启动容器,查看容器卷是否添加成功

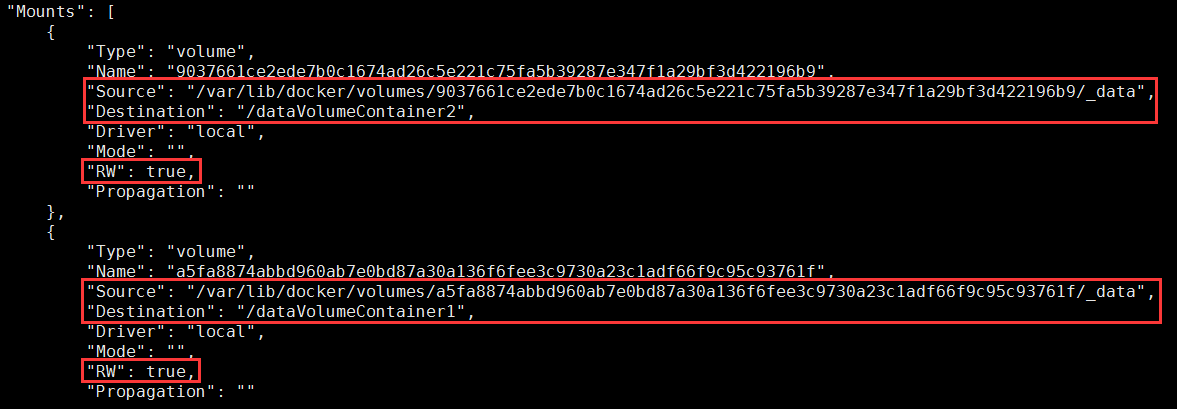


1. 在容器数据卷中创建一个文件,查看是否与宿主机数据共享

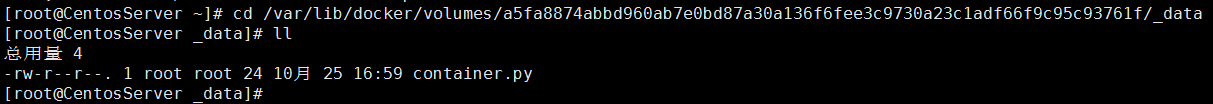


1. Dockerfile文件中虽然未指定宿主机与容器卷的关联关系(但默认已经关联)

查看命令: docker inspect 934bdbe0f417

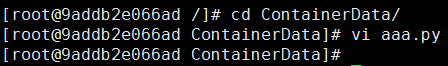


在进入到宿主机中对默认的路径位置进行查看

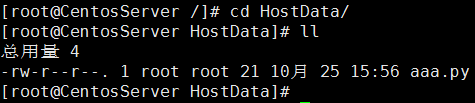


### 容器与宿主机之间数据共享:

**容器Container:**



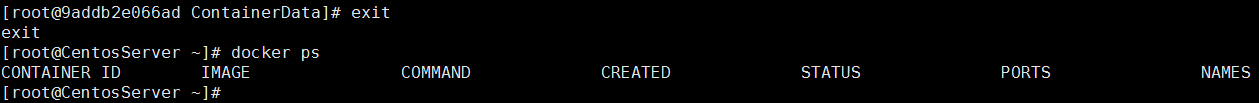
**宿主机Host:**



**注:容器停止并退出,宿主机修改数据后是否共享??? 共享 示例如下:**

**容器Container:**

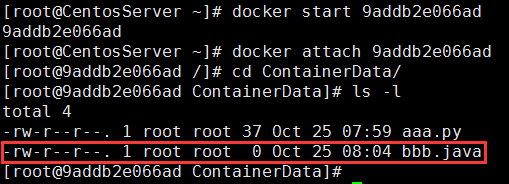
1.容器停止并退出: exit



3.重新启动容器,并进入启动的容器:

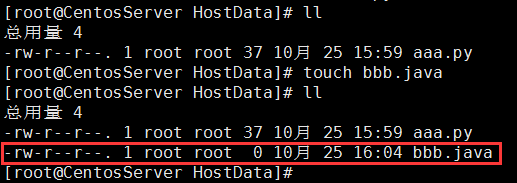
docker start 9addb2e066ad

docker attach 9addb2e066ad



**宿主机Host:**

2.宿主机添加bbb.java文件



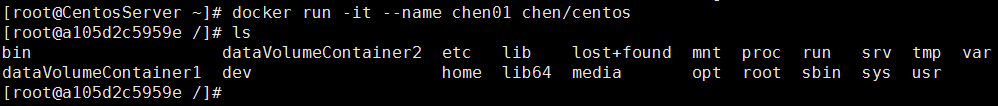
### 容器与容器间数据传递共享:

**结论: 容器之间配置信息的传递,数据卷的生命周期一直持续到没有容器使用它为止**

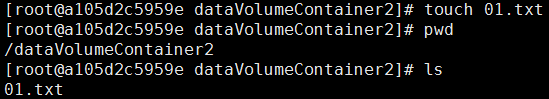
**示例如下:**

1. 启动一个父容器(别名为**chen01**):

docker run -it --name chen01 chen/centos



1. 在父容器数据卷中添加一个文件: touch /dataVolumeContainer2/01.txt



1. 再次启动两个容器(别名分别为**chen02,chen03**),继承父容器chen01: --volumes-from

docker run -it --name **chen02 --volumes-from chen01** chen/centos

docker run -it --name **chen03 --volumes-from chen01** chen/centos



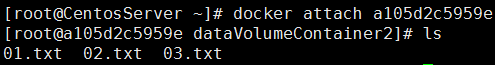


1. 分别在容器chen02和容器chen03中的添加文件:

如chen02 touch /dataVolumeContainer2/02.txt

如chen03 touch /dataVolumeContainer2/03.txt

1. 再次进入如容器(chen01),查看数据卷下的文件是否共享: true



## Docker-Dockerfile构建镜像详解:

### Dockerfile能做什么:

Dockerfile是用来构建Docker镜像的构建文件,是由一系列命令和参数构成的脚本文件

Dockerfile构建步骤:

1. 编写Dockerfile文件: vi /xxx/**Dockerfile**
2. 构建新的镜像文件: **docker build** -f Dockerfile文件路径 -t 新镜像名称:TAG标签 .

如: docker build - Dockerfile - chen/centos .

1. 运行新镜像的容器: **docker run** -it chen/centos

### Dockerfile构建解析:

#### Dockerfile基础知识:

1.每条保留字指令都必须为大写字母且后面要跟随至少一个参数

2.每条保留字指令按照从上到下,顺序执行

3.#表示Dockerfile文件注释内容

4.每条保留字指令都会创建一个新的镜像层,并对镜像进行提交

#### Dockerfile执行流程:

1.Docker基于基础镜像运行一个容器

2.执行一条保留字指令并对容器作出修改

3.执行类似docker commit的操作提交一个新的镜像层

4.Docker再基于刚提交的镜像运行一个新容器

5.执行Dockerfile中的下一条保留字指令直到所有指令都执行完成

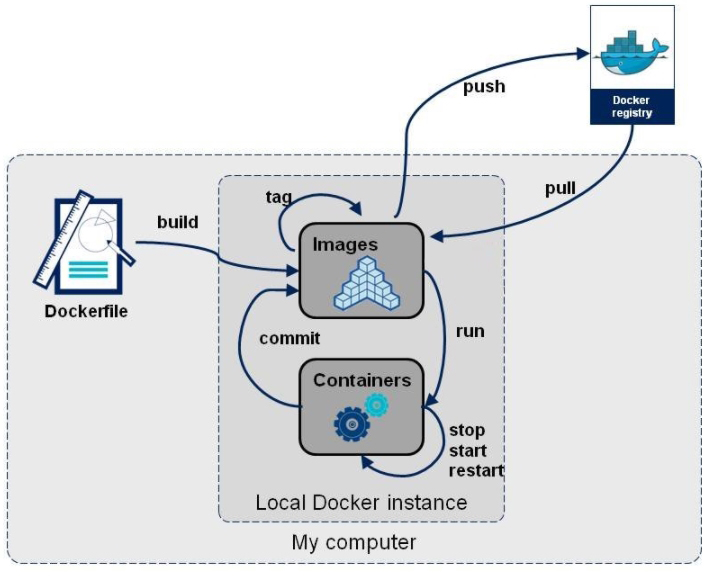
### Dockerfile体系结构:

#### Dockerfile保留字指令:

**Docker的镜像基本上都是通过Base(scratch)镜像安装和配置需要的软件构建出来的**

|  |  |
| --- | --- |
| FROM | (from): 基础镜像,当前构建的新镜像是基于哪个镜像的 |
| MAINTAINER | (maintainer): 新镜像维护者的姓名和邮箱地址 |
| RUN | (run): 容器构建时需要运行的命令 |
| EXPOSE | (expose): 容器运行时对外暴露的端口 |
| WORKDIR | (workdir): 容器创建后,终端默认登陆的进来工作目录路径,一个落脚点 |
| ENV | (env): 构建镜像过程中设置环境变量 |
| ADD | (add): 将宿主机目录下的文件拷贝进镜像且ADD命令会自动处理URL和解压缩tar包 |
| COPY | (copy): 类似ADD,将宿主机目录下的文件拷贝文件和目录到镜像中  将从构建上下文目录中<源路径>的文件/目录复制到新的一层的镜像内的<目标路径>位置  COPY src dest 或 COPY ["src", "dest"] |
| VOLUME | (volume): 容器数据卷,用于数据保存和持久化工作 |
| CMD | (cmd): 指定一个启动时要运行的命令  Dockerfile中可以有多个 CMD 指令,但只有最后一个生效,CMD会被 docker run 之后的参数**覆盖** |
| ENTRYPOINT | (entrypoint): 指定一个容器启动时要运行的命令  docker run之后的参数会当做参数传递给ENTRYPOINT保留字指令(即**追加**),形成新的命令组合 |
| ONBUILD | (onbulid): 当构建一个被继承的Dockerfile时运行命令,父镜像在被子继承后父镜像的onbuild被触发 |
|  |  |

## Docker-Dockerfile自定义镜像:



### 自定义镜像示例一: Dockerfile保留字指令

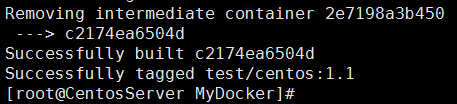
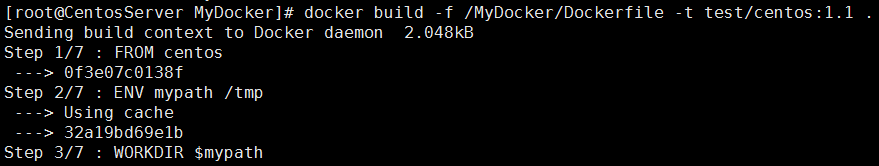
1. **编写: vi /MyDocker/Dockerfile**

**注: 注释内容不能和保留字指令在同一行**

|  |
| --- |
| # FROM继承本地的centos镜像  FROM centos  # 启动容器后,终端的所在目录  #ORKDIR /tmp  ENV mypath /tmp  WORKDIR $mypath  # vim 文本文件编辑  RUN yum -y install vim  # ifconfig查看  RUN yum -y install net-tools  # 暴露出端口号  EXPOSE 80  CMD /bin/bash |

1. **构建: docker build -f Dockerfile -t 新镜像名称:TAG标签 .**

如: docker build -f /MyDocker/Dockerfile -t test/centos:1.1 .



1. **运行: docker run -it新镜像名称:TAG标签**

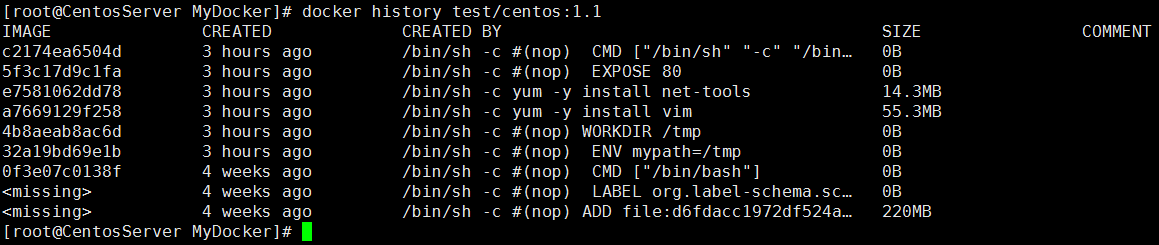
如: docker run -it test/centos:1.1





1. **列出镜像的变更历史: docker history 镜像名称/镜像ID**

如: docker history test/centos:1.1



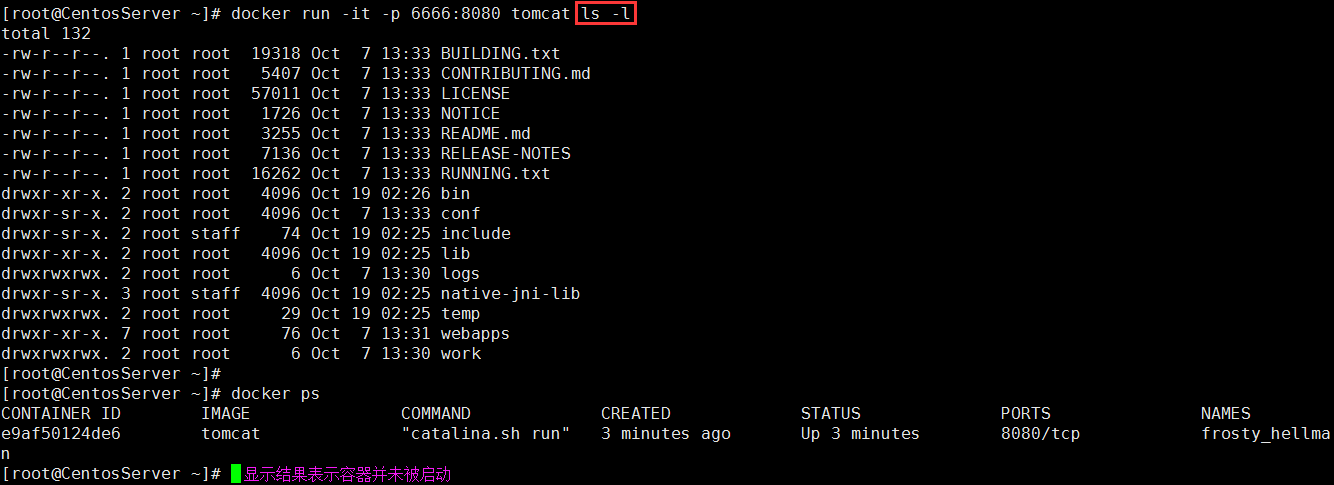
### 自定义镜像示例二: CMD与ENTRYPOINT

#### CMD保留字指令: 覆盖

**指定一个容器启动时要运行的命令,Dockerfile中可以有多个CMD指令,但只有最后一个生效,CMD会被docker run之后的参数替换,如下所示:**

命令: docker run -it -p 6666:8080 tomcat ls -l

如图所示: ls -l参数命令覆盖了Dockerfile中CMD指令: catalina.sh run



|  |  |
| --- | --- |
| # 编辑: vi dockerfile2   |  | | --- | | FROM centos  RUN yum -y install curl  **CMD curl -s https://ip.cn** |   # 构建: docker build -f /MyDocker/dockerfile2 -t centosip .    # 运行: docker run -it centosip |

#### ENTRYPOINT保留字指令: 追加

**指定一个启动时要运行的命令,docker run 之后的参数会当做参数传递给ENTRYPOINT保留字指令(即追加),形成新的命令组合,如 下所示:**

|  |  |
| --- | --- |
| # 编辑: vi dockerfile4   |  | | --- | | FROM centos  RUN yum -y install curl  **ENTRYPOINT ["curl"," -s"," https://ip.cn"]** |   # 构建: docker build -f /MyDocker/dockerfile4 -t centosip4 .    # 运行: docker run -it centosip4 |

### 自定义镜像示例三: ONBUILD

保留字指令: ONBUILD

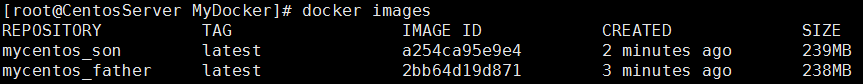
#### 构建父镜像: MyCentOS\_Father

|  |  |
| --- | --- |
| # 编辑: vi dockerfile\_father   |  | | --- | | FROM **centos**  RUN yum -y install curl  ENTRYPOINT ["curl"," -s"," https://ip.cn"]  **ONBUILD** RUN echo "father image onbuild ... 886" |   # 构建: docker build -f /MyDocker/dockerfile\_father -t MyCentOS\_Father .    # 运行: |

#### 构建父镜像: MyCentOS\_Son

|  |  |
| --- | --- |
| # 编辑: vi dockerfile\_son   |  | | --- | | FROM **mycentos\_father**  RUN yum -y install curl  ENTRYPOINT ["curl"," -s"," https://ip.cn"] |   # 构建: docker build -f /MyDocker/dockerfile\_son -t mycentos\_son .    # 运行: |





### 自定义镜像示例四: ADD与COPY

**构建自己的Tomcat8服务:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **# 创建目录: mkdir -p /workspace/tomcat8**  **# 新建文件: touch /workspace/tomcat/aaa.txt**  **# 将jdk8和tomcat8安装压缩包拷贝到该目录: 如下图示**    **# 在/workspace/tomcat8/目录下新建Dockerfile文件并添加如下内容: vi Dockerfile**   |  | | --- | | FROM centos  # 新镜像的维护者姓名  MAINTAINER chen  # 把宿主机当前目录下的aaa.txt拷贝到容器/usr/local/路径下并重新命名  COPY aaa.txt /usr/local/cincontainer.txt  # 把java8与tomcat8安装压缩包文件添加到容器中  ADD apache-tomcat-8.5.47.tar.gz /usr/local/  ADD jdk-8u201-linux-x64.tar.gz /usr/local/  # 安装vim编辑器  RUN yum -y install vim  # 设置工作访问时候的WORKDIR路径,登录落脚点  ENV MYPATH /usr/local  WORKDIR $MYPATH  # 配置java8与tomcat8环境变量  ENV JAVA\_HOME /usr/local/jdk1.8.0\_201  ENV CLASSPATH $JAVA\_HOME/lib/dt.jar:$JAVA\_HOME/lib/tools.jar  ENV CATALINA\_HOME /usr/local/apache-tomcat-8.5.47  ENV CATALINA\_BASE /usr/local/apache-tomcat-8.5.47  ENV PATH $PATH:$JAVA\_HOME/bin:$CATALINA\_HOME/lib:$CATALINA\_HOME/bin  # 容器运行时监听的端口  EXPOSE 8080  # 启动时运行tomcat  # ENTRYPOINT ["/usr/local/apache-tomcat-8.5.47/bin/startup.sh" ]  # CMD ["/usr/local/apache-tomcat-8.5.47/bin/catalina.sh","run"]  CMD /usr/local/apache-tomcat-8.5.47/bin/startup.sh && tail -F /usr/local/apache-tomcat-8.5.47/bin/logs/catalina.out |     **# 构建新镜像: docker build -f /workspace/tomcat8/Dockerfile -t chen/tomcat8 .**  如果: Dockerfile在当前目录下,可简写如: docker build -t chen/tomcat8 .      **# 运行新容器: 挂载宿主机目录运行容器**   |  | | --- | | docker run **-d** **-p** 8888:8080 **--name** mytomcat8 **-v** /workspace/tomcat8/test:/usr/local/apache-tomcat-8.5.47/webapps/test **-v** /workspace/tomcat8/logs/:/usr/local/apache-tomcat-8.5.47/logs/ **--privileged=true** chen/tomcat8 |   **参数解析:**  -d: 以守护进程的方式开启容器  -p: 8888:8080 (p小写),指定宿主机端口号与镜像的容器端口号  --name: 为开启的容器指定一个名字  -v: 容器数据卷,宿主机的绝对路径目录:容器内的绝对路径目录  即: test是tomcat发布的项目,logs目录下存放的tomcat运行项目的日志信息文件  Docker挂载主机目录,Docker访问出现cannot open directory .: Permission denied  解决办法：在挂载目录后多加一个--privileged=true参数即可    访问验证: <http://宿主机IP:设置的端口> ---> 即: http://localhost:8888  注: 当容器是以守护进程的方式(-d)开启的,要想访问容器内的目录或文件,使用exec命令  如: docker exec b2590cc2a8f2 java -ersion    **# 验证是否成功: 进入到test工程项目目录**  宿主机: mkdir WEB-INF  vi web.xml   |  | | --- | | <?xml version="1.0" encoding="UTF-8"?>  <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns="http://java.sun.com/xml/ns/javaee"  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"  id="WebApp\_ID" version="2.5">    <display-name>test</display-name>    </web-app> |   vi a.jsp   |  | | --- | | <%@ page language="java" contentType="text/html; charset=UTF-8"  pageEncoding="UTF-8"%>  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  <html xmlns="http://www.w3.org/1999/xhtml">  <head id="Head1">  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />  </head>  <body>  ---welcome docker tomcat8 test---  <%="i am in docker tomcat self"%>  <br/>  <% System.out.println("---> docker tomcat self ."); %>  </body>  </html> |   查看容器卷: docker exec b2590cc2a8f2 ls -l /usr/local/apache-tomcat-8.5.47/webapps/test    上图所示: 表示在宿主机中test目录下添加的文件共享到容器内的test目录下了  重启容器,查看web工程test发布情况      查看日志信息: cat /workspace/tomcat8/logs/catalina.out |

## Docker上传镜像到云服务器: 如 阿里云

### 登陆阿里云镜像容器服务:

https://cr.console.aliyun.com/cn-shanghai/instances/repositories

### 新镜像的生成方式: 两种方式

**使用Dockerfile文件" docker build -f Dockerfile -t 新镜像名称:TAG标签 . "构建新的镜像文件**

**docker commit -a="提交的镜像作者" -m="提交的镜像说明" 容器名称/容器ID 新镜像名称:TAG标签**

### 将新镜像推送到阿里云仓库:

**阿里云镜像服务中创建仓库:**

创建命名空间名称(即镜像前缀): cd\*\*521\*\*\*\*

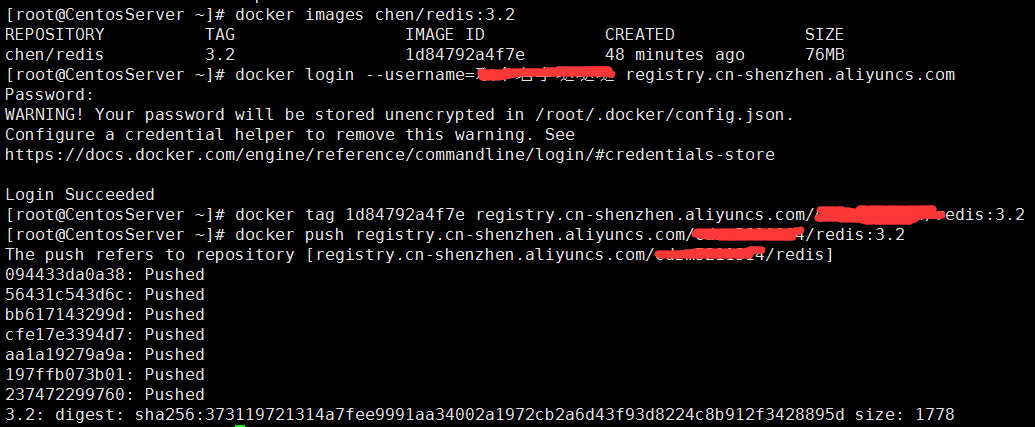
创建镜像仓库名称(即镜像名称): 如 tomcat8 mysql redis

**# 推送镜像到阿里云仓库:**

docker login --username=**阿里云账号** registry.cn-shenzhen.aliyuncs.com

docker tag **[ImageId]** registry.cn-shenzhen.aliyuncs.com/**cd\*\*521\*\*\*\*/tomcat**:**[镜像版本号]**

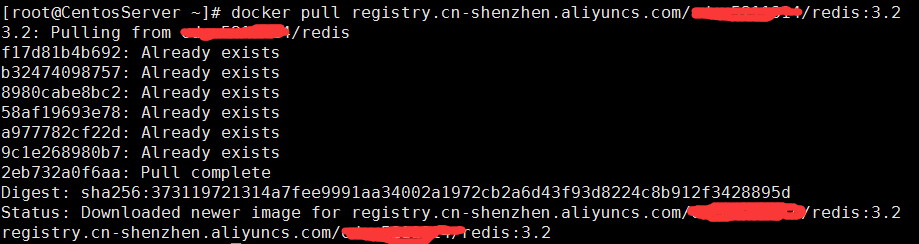
docker push registry.cn-shenzhen.aliyuncs.com/**cd\*\*521\*\*\*\*/tomcat**:**[镜像版本号]**



### 将阿里云镜像拉取到本地:

**#从阿里云仓库拉取镜像:**

docker pull registry.cn-shenzhen.aliyuncs.com/cd\*\*521\*\*\*\*/redis:**[镜像版本号]**



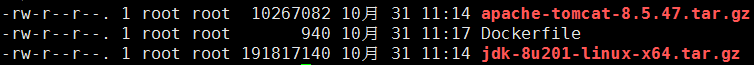
## CentOS7.2-Docker常用安装服务:

### 安装Tmcat8镜像服务:

#### 创建宿主机与容器卷目录:

|  |
| --- |
| mkdir -p /workspace/cdtomcat/conf  mkdir -p /workspace/cdtomcat/logs  mkdir -p /workspace/cdtomcat/webapps |

#### 准备tomcat与jdk压缩包:



#### 编辑Dockerfile文件:

|  |
| --- |
| FROM centos  # 新镜像的维护者姓名  MAINTAINER chen<cdzm5211314@163.com>  # 把java8与tomcat8安装压缩包文件添加到容器中  ADD apache-tomcat-8.5.47.tar.gz /usr/local/  ADD jdk-8u201-linux-x64.tar.gz /usr/local/  # 设置工作访问时候的WORKDIR路径,登录落脚点  ENV MYPATH /usr/local  WORKDIR $MYPATH  # 配置java8与tomcat8环境变量  ENV JAVA\_HOME /usr/local/jdk1.8.0\_201  ENV CLASSPATH $JAVA\_HOME/lib/dt.jar:$JAVA\_HOME/lib/tools.jar  ENV CATALINA\_HOME /usr/local/apache-tomcat-8.5.47  ENV CATALINA\_BASE /usr/local/apache-tomcat-8.5.47  ENV PATH $PATH:$JAVA\_HOME/bin:$CATALINA\_HOME/lib:$CATALINA\_HOME/bin  # 容器运行时监听的端口  EXPOSE 8080  # 启动时运行tomcat  # ENTRYPOINT ["/usr/local/apache-tomcat-8.5.47/bin/startup.sh" ]  # CMD ["/usr/local/apache-tomcat-8.5.47/bin/catalina.sh","run"]  CMD /usr/local/apache-tomcat-8.5.47/bin/startup.sh && tail -F /usr/local/apache-tomcat-8.5.47/bin/logs/catalina.out |

#### 构建新的镜像文件:

命令: docker build -f /workspace/cdtomcat/Dockerfile -t chen/tomcat8 .

#### 运行新容器: 挂载宿主机目录与容器卷目录关系

命令如下:

|  |
| --- |
| docker run -d -p 8080:8080 --name mytomcat8 -v /workspace/cdtomcat/webapps/:/usr/local/apache-tomcat-8.5.47/webapps/ -v /workspace/cdtomcat/logs/:/usr/local/apache-tomcat-8.5.47/logs/ --privileged=true chen/tomcat8 |

### 安装MySQL5.7镜像服务:

#### 拉取5.7版本mysql镜像:

命令: docker pull mysql:5.7

#### 创建宿主机与容器卷目录:

|  |
| --- |
| mkdir -p /workspace/cdmysql/conf  mkdir -p /workspace/cdmysql/logs  mkdir -p /workspace/cdmysql/data |

#### 运行新容器:挂载宿主机目录与容器目录关系

|  |
| --- |
| docker run -d -p 3306:3306 --name mymysql5 -v /workspace/cdmysql/conf:/etc/mysql/conf.d -v /workspace/cdmysql/logs:/logs -v /workspace/cdmysql/data:/var/lib/mysql -e MYSQL\_ROOT\_PASSWORD=root mysql:5.7  **命令参数说明:**  -d : 守护式进程开启容器(后台运行)  -p 3306:3306 :将宿主机的3306端口映射到docker容器的3306端口  --name mymysql5 :为运行的容器取个名称  -v /workspace/cdmysql/conf:/etc/mysql/conf.d :将宿主机/workspace/cdmysql/conf目录下的conf/my.cnf 挂载到容器的 /etc/mysql/conf.d  -v /workspace/cdmysql/logs:/logs :将宿主机/workspace/cdmysql/logs目录挂载到容器的/logs目录  -v /workspace/cdmysql/data:/var/lib/mysql :将宿主机/workspace/cdmysql/data目录挂载到容器的/var/lib/mysql目录  -e MYSQL\_ROOT\_PASSWORD=123456 :初始化 root 用户的密码 |

#### 进入容器交互终端操作mysql:

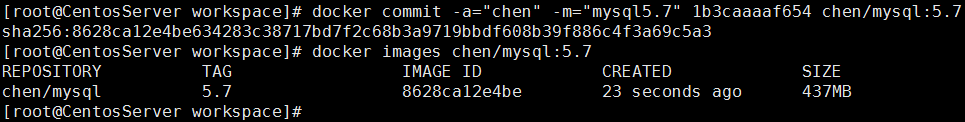
命令: docker exec -it 容器名称/容器ID /bin/bash

#### 宿主机上备份数据库:

命令: docker exec 容器名称/容器ID sh -c ' exec mysqldump --all-databases -uroot -p"123456" ' > /workspace/cdmysql/all-databases.sql

#### 使用容器副本生成新镜像:

docker commit -a="提交的镜像作者" -m="提交的镜像说明" 容器名称/容器ID 新镜像名称:TAG标签



### 安装Redis3.2镜像服务:

#### 拉取3.2版本redis镜像:

命令: docker pull redis:3.2

#### 创建宿主机与容器卷目录:

|  |
| --- |
| mkdir -p /workspace/cdredis/conf  mkdir -p /workspace/cdredis/data  mkdir -p /workspace/cdredis/logs |

#### 运行新容器:挂载宿主机目录与容器目录关系

|  |
| --- |
| docker run -d -p 6379:6379 --name myredis -v /workspace/cdredis/conf:/usr/local/etc/redis/redis.conf -v /workspace/cdredis/data:/data redis:3.2 redis-server /usr/local/etc/redis/redis.conf --appendonly yes |

#### 宿主机下创建redis.conf文件

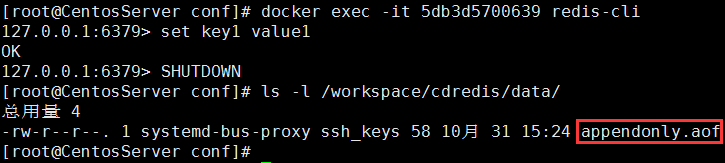
宿主机/xxx/conf/redis.conf目录下新建redis.conf文件

vim /xxx/conf /redis.conf 添加redis的配置文件信息内容

#### 测试redis-cli链接redis服务:

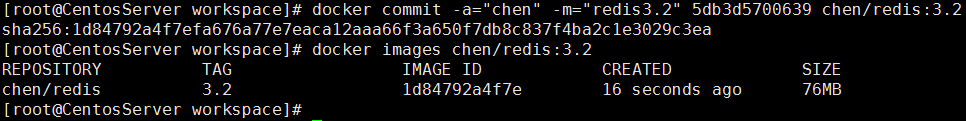
命令: docker exec -it 容器名称/容器ID redis-cli

#### 测试redis持久化文件生成:



#### 使用容器副本生成新镜像:

docker commit -a="提交的镜像作者" -m="提交的镜像说明" 容器名称/容器ID 新镜像名称:TAG标签



# #####################################

**CentOS7.2宿主机: 创建目录**

mkdir -p /workspace/cdtomcat/conf

mkdir -p /workspace/cdtomcat/web

mkdir -p /workspace/cdtomcat/logs

mkdir -p /workspace/cdmysql/conf

mkdir -p /workspace/cdmysql/data

mkdir -p /workspace/cdmysql/logs

mkdir -p /workspace/cdnginx/conf

mkdir -p /workspace/cdnginx/data

mkdir -p /workspace/cdnginx/logs

mkdir -p /workspace/cdredis/conf

mkdir -p /workspace/cdredis/data

mkdir -p /workspace/cdredis/logs

mkdir -p /workspace/cdmongodb/conf

mkdir -p /workspace/cdmongodb/data

mkdir -p /workspace/cdmongodb/logs