# 基础概念

## 概述

Docker是一个可以快速的创建和模拟各类环境的一个工具。Docker能够自动执行重复性任务，例如搭建和配置开发环境。

**Docker不受应用、语言或技术栈限制**

构建、测试、调试和部署以任何编程语言编写的 Linux 和 Windows Server 容器应用，无需担心任何不兼容或版本冲突。

**绝佳的开发体验**

工作就绪时间缩短 65%：快速构建、测试和运行复杂的多容器应用，无需再浪费时间在服务器和开发人员机器上安装和维护软件。所有依赖资源都在容器中运行，消除“在我的机器上可正常工作”的问题。

**内置容器编排**

Docker 内置易于配置的 Swarm 集群功能。在使用最小设置的模拟生产环境中测试和调试应用。

**快速扩展**

内置编排能够扩展到数千个节点和容器。Docker 容器能够在短短数秒之内启动和停止，便于扩展应用服务，以满足客户的高峰需求，并在峰值下降时缩减规模。

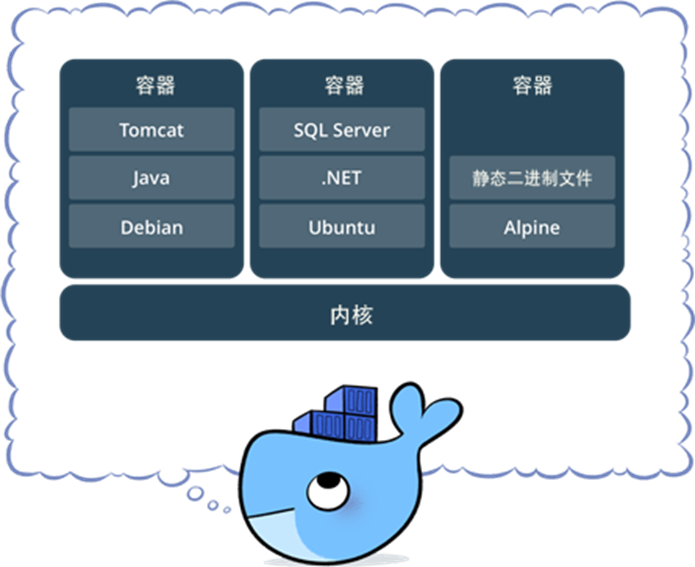
**提高 IT 效率**

Docker 让客户轻松部署、识别和解决问题，降低总体 IT 运维成本。缩短部署更新的停机时间，或者迅速回滚，尽量减少中断运行情况。

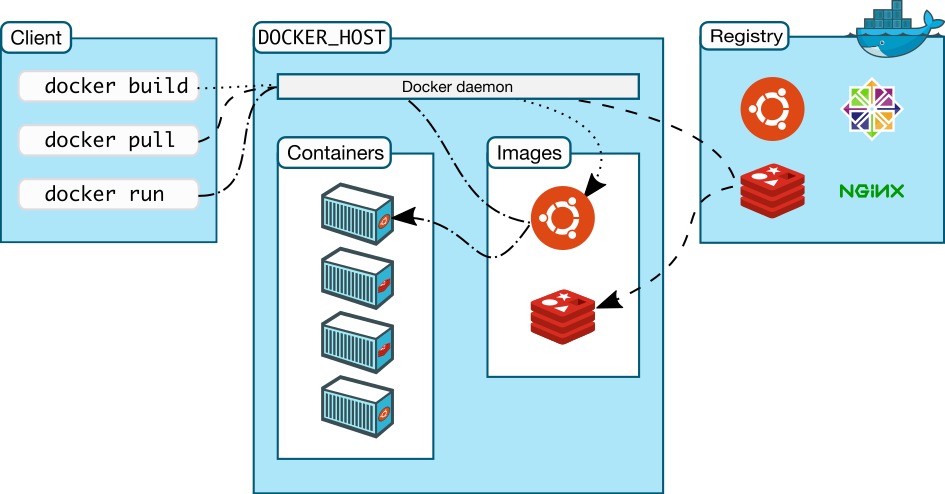
**轻松共享应用**

Docker 确保应用在任何环境中都能始终如一地工作。在 Docker 镜像中，整个技术栈和配置都是镜像的一部分，用户只需安装 Docker，无需配置主机环境。

## 组成部分



对于Docker而言，主要是使用了容器技术。有了容器，就可以将软件运行所需的所有资源打包到一个隔离的容器中。容器与虚拟机不同，不需要捆绑一整套操作系统，只需要软件工作所需的库资源和设置。系统因此而变得高效、轻量、自给自足，还能保证部署在任何环境中的软件都能始终如一地运行。



### 客户端和服务端

Docker是一个（C/S）架构的程序。Docker客户端只需向Docker服务器或者守护进程发出请求，服务器或者守护进程将完成所有的工作并返回结果。Docker守护进程有时也称为Docker引擎。

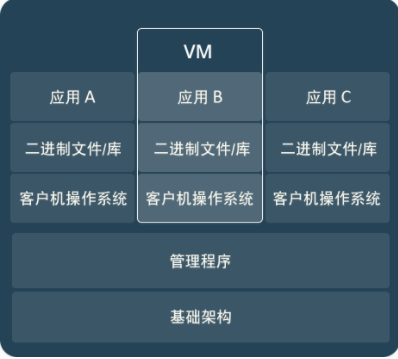
### 镜像（Images）

镜像就是程序运行的环境的只读版本。其包含了所有程序的依赖软件和配置。

### 容器（Container）

Docker 利用容器（Container）来运行应用。容器是从镜像创建的运行实例。它可以被启动、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。

**容器和虚拟机比较**

**容器**是一个应用层抽象，用于将代码和依赖资源打包在一起，多个容器可以在同一台机器上运行，共享操作系统内核，但各自作为独立的进程在用户空间中运行。与虚拟机相比，容器占用的空间较小（容器镜像大小通常只有几十兆），瞬间就能完成启动。

**虚拟机**是一个物理硬件层抽象，用于将一台服务器变成多台服务器。管理程序允许多个VM在一台机器上运行。每个VM都包含一整台操作系统、一个或多个应用、必要的二进制文件和库资源，因此占用大量空间。而且VM启动也十分缓慢。

### 仓库（Repository）

仓库用来保存镜像，可以理解为代码控制中的代码仓库。

## 应用场景

**微服务**

微服务拆分后，一个项目可能部署包就成倍增加了，而且可能各微服务之间的技术栈是不同的，这时候docker就是最佳选择了。

**持续集成和持续部署 (CI/CD)**

结合Jenkins,通过 Docker 加速应用管道自动化和应用部署，交付速度了有很大程度的提高。

**IT 基础设施优化**

Docker 和容器有助于优化 IT 基础设施的利用率和成本。优化不仅仅是指削减成本，还能确保在适当的时间有效地使用适当的资源。

**容器化传统应用**

容器不仅能提高现有应用的安全性和可移植性，还能节约成本。

# 安装和卸载

现在的Docker已经能支持Windows、Mac及Linux的各大操作系统了。但在生产阶段，应该Linux才是主流。其他的平台还是主要以学习为主。但对于Linux平台而言，又有CentOS、Debian、Fedora、Ubuntu等等不同的发行版。不同的发行版安装过程是稍有不同的。

## 安装Docker

Docker安装命令如下，如果没有-y时，下载完安装包后会出现选择项，是否安装；设置-y后表示默认选择yes。

|  |
| --- |
| [root@\*\*\* ~]# sudo yum [-y] install docker-ce |

安装完成后启动Docker：

|  |
| --- |
| [root@\*\*\* ~]# sudo systemctl start docker |

然后使用如下命令验证Docker是否安装成功：

|  |
| --- |
| [root@\*\*\* ~]# sudo docker info |

出现如下内容后表示Docker安装成功：

|  |
| --- |
| Client:  Debug Mode: false  Server:  Containers: 2  Running: 1  Paused: 0  Stopped: 1  Images: 2  Server Version: 19.03.6 |

由于国内墙的问题，安装完成后还需要设置阿里云加速器，推荐直接使用阿里云的镜像。进入阿里云控制台，控制台首页(产品与服务)–>容器镜像服务—>镜像加速器，URL地址如下：

https://cr.console.aliyun.com/cn-hangzhou/instances/mirrors

然后按如下步骤操作：

1、复制加速器地址；

2、编写文件/etc/docker/daemon.json(如果不存在，手动创建daemon.json文件)，内容为

|  |
| --- |
| [root@\*\*\* ~]# sudo tee /etc/docker/daemon.json <<-'EOF'  {  "registry-mirrors": ["https://镜像地址.mirror.aliyuncs.com"]  }  EOF |

3、重新加载daemon.json文件，然后重启Docker，至此加速器就配置好了。

|  |
| --- |
| [root@\*\*\* ~]# sudo systemctl daemon-reload  [root@\*\*\* ~]# sudo systemctl restart docker |

配置完加速器后，可以测试运行官方提供的hello-world镜像

|  |
| --- |
| [root@\*\*\* ~]# sudo docker run hello-world |

出现如下内容时表示hello-world镜像运行成功。

|  |
| --- |
| Hello from Docker!  This message shows that your installation appears to be working correctly. |

### 安装错误问题

**No package docker-ce available**

这个问题需要安装基础包和设置yum源(由于国内环境，因此直接使用阿里云镜像地址)

**安装基础包**

yum-utils提供yum-config-manager功能，另外两个是devicemapper驱动依赖。

|  |
| --- |
| [root@\*\*\* ~]# sudo yum install -y yum-utils \  device-mapper-persistent-data \  lvm2 |

**设置yum源**

|  |
| --- |
| [root@\*\*\* ~]# sudo yum-config-manager \  --add-repo \  https://download.docker.com/linux/centos/docker-ce.repo |

**https://download.docker.com/linux/centos/repodata/repomd.xml:**

**[Errno 14] HTTPS Error 404 - Not Found**

查看/etc/yum.repos.d目录下的docker-ce.repo文件是否包含https://download-stage.docker.com，替换成http://mirrors.aliyun.com/docker-ce,或目录包含download.docker.com\_linux\_centos\_.repo(未替换阿里云镜像加了官网的源地址时出现)文件的，删除此文件。

### 安装特定版本

使用如下命令查看Docker版本：

|  |
| --- |
| [root@\*\*\* ~]# yum list docker-ce --showduplicates | sort -r |

使用上述命令后会出现如下结果：

|  |
| --- |
| Loading mirror speeds from cached hostfile  Loaded plugins: fastestmirror  Installed Packages  docker-ce.x86\_64 3:19.03.6-3.el7 docker-ce-stable  docker-ce.x86\_64 3:19.03.6-3.el7 @docker-ce-stable  docker-ce.x86\_64 3:19.03.5-3.el7 docker-ce-stable  docker-ce.x86\_64 3:19.03.4-3.el7 docker-ce-stable  docker-ce.x86\_64 3:19.03.3-3.el7 docker-ce-stable  docker-ce.x86\_64 3:19.03.2-3.el7 docker-ce-stable |

如果选定了摸一个版本，则使用如下命令进行安装：

|  |
| --- |
| # yum install <FULLY-QUALIFIED-PACKAGE-NAME> 注意：包名是截取第一列和第二列的部分值组合而成  [root@\*\*\* ~]# sudo yum install docker-ce-18.03.1.ce |

## 卸载Docker

Docker卸载命令如下：

|  |
| --- |
| [root@\*\*\* ~]# sudo yum remove docker \  docker-client \  docker-client-latest \  docker-common \  docker-latest \  docker-latest-logrotate \  docker-logrotate \  docker-selinux \  docker-engine-selinux \  docker-engine |

# 常用命令

可以通过如下命令来查看Dokcer命令清单：

|  |
| --- |
| [root@\*\*\* ~]# docker [help]  Usage: docker [OPTIONS] COMMAND  A self-sufficient runtime for containers  Options:  --config string Location of client config files (default "/root/.docker")  -c, --context string Name of the context to use to connect to the daemon  (overrides DOCKER\_HOST env var and default context set with  "docker context use")  -D, --debug Enable debug mode  -H, --host list Daemon socket(s) to connect to  -l, --log-level string Set the logging level ("debug"|"info"|"warn"|"error"|"fatal")  (default "info")  --tls Use TLS; implied by --tlsverify  --tlscacert string Trust certs signed only by this CA (default  "/root/.docker/ca.pem")  --tlscert string Path to TLS certificate file (default "/root/.docker/cert.pem")  --tlskey string Path to TLS key file (default "/root/.docker/key.pem")  --tlsverify Use TLS and verify the remote  -v, --version Print version information and quit  Management Commands:  builder Manage builds  config Manage Docker configs  container Manage containers  context Manage contexts  engine Manage the docker engine  image Manage images  network Manage networks  node Manage Swarm nodes  plugin Manage plugins  secret Manage Docker secrets  service Manage services  stack Manage Docker stacks  swarm Manage Swarm  system Manage Docker  trust Manage trust on Docker images  volume Manage volumes  Commands:  attach Attach local standard input, output, and error streams to a running container  build Build an image from a Dockerfile  commit Create a new image from a container's changes  cp Copy files/folders between a container and the local filesystem  create Create a new container  diff Inspect changes to files or directories on a container's filesystem  events Get real time events from the server  exec Run a command in a running container  export Export a container's filesystem as a tar archive  history Show the history of an image  images List images  import Import the contents from a tarball to create a filesystem image  info Display system-wide information  inspect Return low-level information on Docker objects  kill Kill one or more running containers  load Load an image from a tar archive or STDIN  login Log in to a Docker registry  logout Log out from a Docker registry  logs Fetch the logs of a container  pause Pause all processes within one or more containers  port List port mappings or a specific mapping for the container  ps List containers  pull Pull an image or a repository from a registry  push Push an image or a repository to a registry  rename Rename a container  restart Restart one or more containers  rm Remove one or more containers  rmi Remove one or more images  run Run a command in a new container  save Save one or more images to a tar archive (streamed to STDOUT by default)  search Search the Docker Hub for images  start Start one or more stopped containers  stats Display a live stream of container(s) resource usage statistics  stop Stop one or more running containers  tag Create a tag TARGET\_IMAGE that refers to SOURCE\_IMAGE  top Display the running processes of a container  unpause Unpause all processes within one or more containers  update Update configuration of one or more containers  version Show the Docker version information  wait Block until one or more containers stop, then print their exit codes  # 对于某个命令想知道其详细的参数选项时，可依照此模式进行查看。  Run 'docker COMMAND --help' for more information on a command. |

## 镜像常用命令

### search搜索镜像

|  |
| --- |
| [root@iZwz9auj7s2xykmtffvuzoZ ~]# docker search hello-world  NAME DESCRIPTION STARS OFFICIAL AUTOMATED  hello-world Hello World! (an… 1143 [OK]  kitematic/hello-… A light-weight… 142  tutum/hello-world Image to… 69 [OK]  dockercloud/hello-… Hello World! 18 [OK]  crccheck/hello-wor… Hello World web se… 12 [OK] |

### pull拉取镜像

|  |
| --- |
| # 用法：docker pull [OPTIONS] NAME[:TAG|@DIGEST]  [root@\*\*\*~]# docker pull hello-world  Using default tag: latest  latest: Pulling from library/hello-world  1b930d010525: Already exists  Digest: sha256:fc6a51919cfeb2e6763f62b6d9e8815acbf7cd2e476ea353743570610737b752  Status: Downloaded newer image for hello-world:latest  docker.io/library/hello-world:latest |

### images查看已下载镜像

|  |
| --- |
| [root@\*\*\* ~]# docker images [imageName]  REPOSITORY TAG IMAGE ID CREATED SIZE  rabbitmq management 985adbf13062 6 days ago 181MB  hello-world latest fce289e99eb9 14 months ago 1.84kB |

### tag镜像拷贝

|  |
| --- |
| # 比如，想创建拷贝一个镜像`hello-wrold`，同时命名为`images/hello-world`  [root@\*\*\* ~]# docker tag hello-world images/hello-world  [root@\*\*\* ~]# docker images  REPOSITORY TAG IMAGE ID CREATED SIZE  rabbitmq management 985adbf13062 6 days ago 181MB  images/hello-world latest fce289e99eb9 14 months ago 1.84kB  hello-world latest fce289e99eb9 14 months ago 1.84kB |

### build基于Dockerfile创建一个新的镜像

|  |
| --- |
| # 使用当前目录下的Dockerfile,同时镜像命名(`-t`，指tag)为：first-image  [root@\*\*\* ~]# docker build -t first-image |

### rmi删除镜像

删除镜像可以根据镜像名称或IMAGE\_ID来删除，当镜像有容器在使用时，是无法删除的，需先删除容器再删除镜像。

|  |
| --- |
| [root@\*\*\* ~]# docker rmi images/hello-world |

根据IMAGE\_ID来删除镜像时，如果有多个相同IMAGE\_ID的镜像，则会报出错误提示：

|  |
| --- |
| [root@\*\*\* ~]# docker rmi fce289e99eb9  Error response from daemon: conflict: unable to delete fce289e99eb9 (must be forced) - image is referenced in multiple repositories |

此时如果想要删除多个相同IMAGE\_ID的镜像，可以使用-f参数强制删除：

|  |
| --- |
| [root@\*\*\*~]# docker rmi -f fce289e99eb9  Untagged: images/hello-world:latest  Untagged: hello-world:latest  Untagged: hello-world@sha256:fc6a51919cfeb2e6763f62b6d9e8815acbf7cd2e476ea353743570610737b752  Deleted: sha256:fce289e99eb9bca977dae136fbe2a82b6b7d4c372474c9235adc1741675f587e |

## 容器常用命令

### run运行容器

|  |
| --- |
| [root@\*\*\* ~]# docker run hello-world |

执行运行容器命令时可选参数如下：

|  |
| --- |
| -a stdin: 指定标准输入输出内容类型，可选 STDIN/STDOUT/STDERR 三项；  -d: 后台运行容器，并返回容器ID；  -i: 以交互模式运行容器，通常与 -t 同时使用；  -p: 端口映射，格式为：主机(宿主)端口:容器端口  -t: 为容器重新分配一个伪输入终端，通常与 -i 同时使用；  --name="nginx-lb": 为容器指定一个名称；  --dns 8.8.8.8: 指定容器使用的DNS服务器，默认和宿主一致；  --dns-search example.com: 指定容器DNS搜索域名，默认和宿主一致；  -h "mars": 指定容器的hostname；  -e username="ritchie": 设置环境变量；  --env-file=[]: 从指定文件读入环境变量；  --cpuset="0-2" or --cpuset="0,1,2": 绑定容器到指定CPU运行；  -m :设置容器使用内存最大值；  --net="bridge": 指定容器的网络连接类型，支持 bridge/host/none/container: 四种类型；  --link=[]: 添加链接到另一个容器；  --expose=[]: 开放一个端口或一组端口； |
|  |

### ps容器列表

使用ps命令，可以列举出当前运行的容器，需要所有容器时，加入-a选项即可。

|  |
| --- |
| [root@iZwz9auj7s2xykmtffvuzoZ ~]# docker ps  CONTAINER ID 33694dfb1402  IMAGE rabbitmq:management  COMMAND "docker-entrypoint.s…"  CREATED 45 hours ago  STATUS Up 45 hours  PORTS 4369/tcp, 5671/tcp, 0.0.0.0:5672->5672/tcp, 15671/tcp, 25672/tcp, 0.0.0.0:15672->15672/tcp NAMES rabbit |

### stop停止容器

|  |
| --- |
| # docker stop 容器ID  [root@\*\*\* ~]# docker stop 3ba5b7475423 |

### start启动已停止容器

|  |
| --- |
| # docker start 容器ID  [root@\*\*\* ~]# docker start 3ba5b7475423 |

### restart重启容器

|  |
| --- |
| # docker restart 容器ID  [root@\*\*\* ~]# docker restart 3ba5b7475423 |

### kill强制停止容器

|  |
| --- |
| # docker kill 容器ID  [root@\*\*\* ~]# docker kill 3ba5b7475423 |

### rm删除容器

|  |
| --- |
| # docker rm 容器ID  [root@\*\*\* ~]# docker rm 3ba5b7475423 |

### exec进入容器

在一些场景下，比如想查看redis的客户端redis-cli时，这个时候就需要进入容器了。进入容器有很多中，这里就exec进行讲解。

|  |
| --- |
| # docker exec -it 容器ID 参数  [root@\*\*\* ~]# docker exec -it 3ba5b7475423 redis-cli  参数说明：  -d：分离模式: 在后台运行  -i：即使没有附加也保持STDIN 打开  -t：分配一个伪终端 |

执行上述命令后进入redis-cli，就可以进行相应的操作了

|  |
| --- |
| [root@xxx ~]# docker exec -it 3ba5b7475423 redis-cli  127.0.0.1:6379> keys \*  (empty list or set)  127.0.0.1:6379> set name okong  OK  127.0.0.1:6379> keys \*  1) "name"  127.0.0.1:6379> get name  "okong"  127.0.0.1:6379> |

### commit容器中创建镜像

在制作一些私有镜像时，常常是依赖于一个基础镜像后，然后进入容器中进行相关系统环境的配置，或者相应的优化。但若容器一删除，之前的修改都会没有了。故在这些场景下，可直接从修改后的容器中创建一个自己的私有镜像，这样里面的一些环境和相关优化项还是保留的。

|  |
| --- |
| # docker commit [options] 容器id name:tag  [root@\*\*\* ~]# docker commit 3ba5b7475423 lqdev.cn/redis:1  参数说明：  -a：提交的镜像作者  -c：使用Dockerfile指令来创建镜像  -m：提交时的说明文字  -p：在commit时，将容器暂停 |

## 其他常用命令

### logs查看日志

|  |
| --- |
| # docker logs [OPTIONS] 容器ID  [root@xxx ~]# docker logs -f 3ba5b7475423  参数说明：  -f : 跟踪日志输出  --since :显示某个开始时间的所有日志  -t : 显示时间戳  --tail :仅列出最新N条容器日志 |

### cp宿主和容器之间相互拷贝文件

|  |
| --- |
| # docker cp 容器名：要拷贝的文件在容器里面的路径 要拷贝到宿主机的相应路径 # docker cp 要拷贝的文件路径 容器名：要拷贝到容器里面对应的路径 [root@\*\*\* ~]# docker cp 3ba5b7475423:/opt/a.json /opt [root@\*\*\* ~]# docker cp /opt/a.json 3ba5b7475423:/opt |

# Dockerfile

Dockerfile是一个文本文件，里面包含了若干条指令，每条指令描述了构建镜像的细节。简单来说，它就是由一系列指令和参数构成的脚本文件，从而构建出一个新的镜像文件。

## 简单示例

首先编写一个简单的Dockerfile文件，然后上传到Linux服务器的/opt/docker目录下

|  |
| --- |
| FROM nginx LABEL author="作者：xmm" LABEL version="版本：v0.1" LABEL desc="说明：修改nginx首页提示" RUN echo 'hello,oKong' > /usr/share/nginx/html/index.html CMD ["nginx", "-g", "daemon off;"] |

然后执行如下docker build -t IMAGE\_NAME 命令构建docker镜像，注意最后有个 . 参数，表示是当前路径，然后控制台会显示每一步具体的执行说明。

|  |
| --- |
| [root@iZwz9auj7s2xykmtffvuzoZ docker]# docker build -t my-image .  Sending build context to Docker daemon 2.048kB  Step 1/6 : FROM nginx  latest: Pulling from library/nginx  68ced04f60ab: Pull complete  28252775b295: Pull complete  a616aa3b0bf2: Pull complete  Digest: sha256:2539d4344dd18e1df02be842ffc435f8e1f699cfc55516e2cf2cb16b7a9aea0b  Status: Downloaded newer image for nginx:latest  ---> 6678c7c2e56c  Step 2/6 : LABEL author="作者：oKong"  ---> Running in c53881205f8b  Removing intermediate container c53881205f8b  ---> b0859e0b2b12  Step 3/6 : LABEL version="版本：v0.1"  ---> Running in a2eb7585838d  Removing intermediate container a2eb7585838d  ---> d70c028c1eb3  Step 4/6 : LABEL desc="说明：修改nginx首页提示"  ---> Running in 316952dce592  Removing intermediate container 316952dce592  ---> 971c638f9cfb  Step 5/6 : RUN echo 'hello,oKong' > /usr/share/nginx/html/index.html  ---> Running in beed5ad67d7d  Removing intermediate container beed5ad67d7d  ---> 0d3f414e60b3  Step 6/6 : CMD ["nginx", "-g", "daemon off;"]  ---> Running in 83dd57cb536c  Removing intermediate container 83dd57cb536c  ---> eb94d7a83b93  Successfully built eb94d7a83b93  Successfully tagged my-image:latest |

然后执行docker images命令就可以看到刚刚使用Dockerfile文件构建出来的docker镜像了。

|  |
| --- |
| [root@iZwz9auj7s2xykmtffvuzoZ docker]# docker images  REPOSITORY TAG IMAGE ID CREATED SIZE  my-image latest eb94d7a83b93 7 minutes ago 127MB  nginx latest 6678c7c2e56c 4 days ago 127MB |

## 命令说明

### FROM

指定基础镜像，放在第一行，其格式如下

|  |
| --- |
| FROM <image>  FROM <image>:<tag> FROM <image>:<digest> |

若想构建一个最小的镜像，不想基于其他任何镜像时，可使用如下方式

|  |
| --- |
| FROM scratch |

### LABEL

镜像元数据，可以设置镜像的任何元数据，格式如下：

|  |
| --- |
| LABEL <key>=<value> <key>=<value> <key>=<value> ... # 或  LABEL <key>=<value>  LABEL <key>=<value>  LABEL <key>=<value>  ... |

然后利用docker inspect [IMAGE\_NAME|IMAGE\_ID]命令进行查看。

|  |
| --- |
| [root@iZwz9auj7s2xykmtffvuzoZ opt]# docker inspect my-image  …  "Labels": {  "author": "作者：xmm",  "desc": "说明：修改nginx首页提示",  "maintainer": "NGINX Docker Maintainers <docker-maint@nginx.com>",  "version": "版本：v0.1"  },  … |

### ENV

设置环境变量，之后的命令都可以用此变量进行赋值，格式如下：

|  |
| --- |
| ENV <key> <value> ENV <key>=<value> ... |

### VOLUME

定义匿名卷，创建挂载点，即向基于所构建镜像创始的容器添加卷，一个卷可以存在于一个或多个容器的指定目录，该目录可以绕过联合文件系统，并具有以下功能：

卷可以容器间共享和重用

容器并不一定要和其它容器共享卷

修改卷后会立即生效

对卷的修改不会对镜像产生影响

卷会一直存在，直到没有任何容器在使用它

VOLUME可以将源代码、数据或其它内容添加到镜像中，而又不并提交到镜像中，并使多个容器间共享这些内容。

|  |
| --- |
| VOLUME ["/data"] |

### COPY

复制文件，主要就是构建镜像时，进行拷贝文件到镜像的指定路径下，格式为：

|  |
| --- |
| COPY <源路径>... <目标路径> COPY ["<源路径1>",... "<目标路径>"] |

### ADD

更高级的复制文件，ADD指令和COPY的格式和性质基本一致。但是在COPY基础上增加了一些功能。比如<源路径>可以是一个 URL，这种情况下，Docker引擎会试图去下载这个链接的文件放到<目标路径>去。

### EXPOSE

为镜像设置监听端口，容器运行时会监听改端口，格式为：

|  |
| --- |
| EXPOSE <port> [<port>/<protocol>...]  # 为镜像设置监听端口，同时，也能指定协议名 EXPOSE 80/udp |

### ARG

该命令用于设置构建参数，该参数在容器运行时是获取不到的，只有在构建时才能获取。这也是其和ENV的区别。

|  |
| --- |
| ARG <name>[=<default value>] |

### RUN

在镜像的构建过程中执行特定的命令，并生成一个中间镜像。格式:

|  |
| --- |
| RUN <command> # 或者 RUN ["executable", "param1", "param2"] |

第一种后边直接跟shell命令，在linux操作系统上默认 /bin/sh -c；在windows操作系统上默认 cmd /S /C。

第二种是类似于函数调用，可将executable理解成为可执行文件，后面就是两个参数。

|  |
| --- |
| RUN /bin/bash -c 'source $*HOME*/.bashrc; echo $*HOME* RUN ["/bin/bash", "-c", "echo hello"] |

多行命令不要写多个RUN，原因是Dockerfile中每一个指令都会建立一层.多少个RUN就构建了多少层镜像，会造成镜像的臃肿、多层，不仅仅增加了构件部署的时间，还容易出错。

RUN书写时的换行符是\。

CMD

容器启动时要运行的命令，第一种和第二种都是可执行文件加上参数的形式，第三种是shell写法。

|  |
| --- |
| CMD ["executable","param1","param2"] CMD ["param1","param2"] CMD command param1 param2 |

这里边包括参数的一定要用双引号，就是双引号",不能是单引号。千万不能写成单引号。原因是参数传递后，docker解析的是一个JSON array。

|  |
| --- |
| CMD [ "sh", "-c", "echo $*HOME*" ] CMD [ "echo", "$*HOME*" ] |

### ENTRYPOINT

ENTRYPOINT 用于给容器配置一个可执行程序。也就是说，每次使用镜像创建容器时，通过 ENTRYPOINT 指定的程序都会被设置为默认程序。ENTRYPOINT 有以下两种形式：

|  |
| --- |
| ENTRYPOINT ["executable", "param1", "param2"] ENTRYPOINT command param1 param2 |

ENTRYPOINT 与 CMD 非常类似，不同的是通过docker run执行的命令不会覆盖 ENTRYPOINT，而docker run命令中指定的任何参数，都会被当做参数再次传递给ENTRYPOINT。Dockerfile 中只允许有一个 ENTRYPOINT 命令，多指定时会覆盖前面的设置，而只执行最后的ENTRYPOINT 指令。

docker run运行容器时指定的参数都会被传递给ENTRYPOINT，且会覆盖 CMD 命令指定的参数。如，执行docker run <image> -d时，-d 参数将被传递给入口点。也可以通过docker run --entrypoint重写 ENTRYPOINT 入口点。如：可以像下面这样指定一个容器执行程序：

|  |
| --- |
| ENTRYPOINT ["/usr/bin/nginx"] |

### WORKDIR

用于在容器内设置一个工作目录，通过WORKDIR设置工作目录后，Dockerfile 中其后的命令RUN、CMD、ENTRYPOINT、ADD、COPY等命令都会在该目录下执行。

|  |
| --- |
| WORKDIR /opt/docker/workdir |

### USER

用于指定运行镜像所使用的用户，使用USER指定用户后，Dockerfile 中其后的命令RUN、CMD、ENTRYPOINT都将使用该用户。镜像构建完成后，通过docker run运行容器时，可以通过-u参数来覆盖所指定的用户。

|  |
| --- |
| USER okong |

## 构建自定义镜像