学习资料

<https://blog.csdn.net/zzq900503/article/details/80230415>

<http://c.biancheng.net/view/3118.html>

<https://www.cnblogs.com/sparkdev/tag/docker/>

<https://blog.csdn.net/a745233700/article/details/80431045>

大牛博客

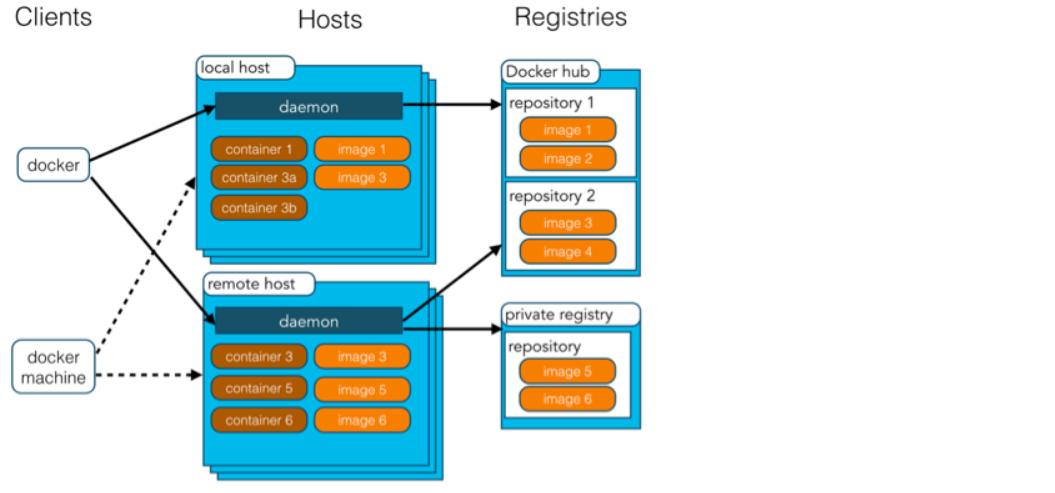
<https://blog.51cto.com/lizhenliang>

## 一 Docker介绍

容器、沙箱

docker如VM，image就是你装vm虚拟机时的iso文件，容器就是正在vm中运行的虚拟机

### 1 Docker架构



|  |
| --- |
| Docker 守护进程(daemon)  Docker Daemon是Docker架构中运行在后台的守护进程，Docker daemon运行在宿主机上，用户不能直接操作daemon，只能通过docker client转达。  Docker 镜像(Images)  Docker 镜像是用于创建 Docker 容器的模板。  Docker 容器(Container)  容器是独立运行的一个或一组应用。  Docker 客户端(Client)  Docker 客户端通过命令行或者其他工具使用 Docker API (https://docs.docker.com/reference/api/docker\_remote\_api) 与 Docker 的守护进程通信。  Docker 主机(Host)  一个物理或者虚拟的机器用于执行 Docker 守护进程和容器。  Docker 仓库(Registry)  Docker 仓库用来保存镜像，可以理解为代码控制中的代码仓库。  Docker Hub(https://hub.docker.com) 提供了庞大的镜像集合供使用。  Docker Machine  Docker Machine是一个简化Docker安装的命令行工具，通过一个简单的命令行即可在相应的平台上安装Docker，比如VirtualBox、 Digital Ocean、Microsoft Azure。 |

### **2 Docker**镜像

镜像，三大核心组件之一，只读的文件系统。运行容器时，需要本地有对应容器的镜像文件。如果没有，则从远程仓库中获取该镜像。

镜像是一个只读的层（layer），由文件系统叠加构成。而镜像又是容器(Container)的构成单元，我们一般会将应用构建成标准的镜像组件，一或多个镜像叠加又构成了容器。

在Docker中将这种文件称为镜像（Image），或者说上Docker容器的文件系统是由多个镜像来构成的。一个镜像，可以放在另一个镜像之上。位于底层的镜像称之为父镜像，而位于最底层的镜像称之为基础镜像（Base Image）。

当从一个镜像启动容器时，Docker会从该镜像的最顶层加载一个读／写文件系统，而我们的应用就会运行于这个读/写层中。

### 3 Docker容器

类似linux系统环境，运行和隔离应用。容器从镜像启动的时候，docker会在镜像的最上一层创建一个可写层，镜像本身是只读的，保持不变。

容器是镜像的一个运行实例，不同的是它带有额外的可写层。

可认为docker容器就是独立运行的一个或一组应用，以及它们所运行的必需环境。

### 4 Docker仓库

仓库是集中存放镜像的地方。每个服务器上可以有多个仓库。

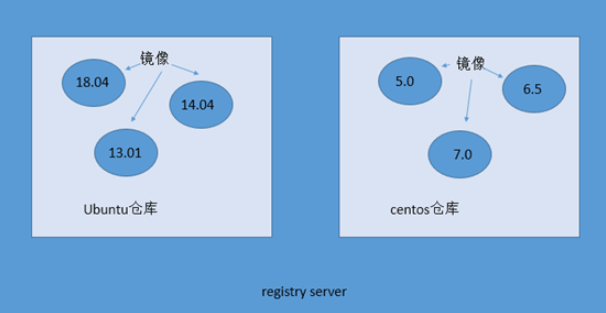
仓库又分为公有仓库（DockerHub、dockerpool）和私有仓库

公有库：

DockerHub：docker官方维护的一个公共仓库https://hub.docker.com，其中包括了15000多个的镜像，大部分都可以通过dockerhub直接下载镜像。也可通过docker search和docker pull命令来下载。

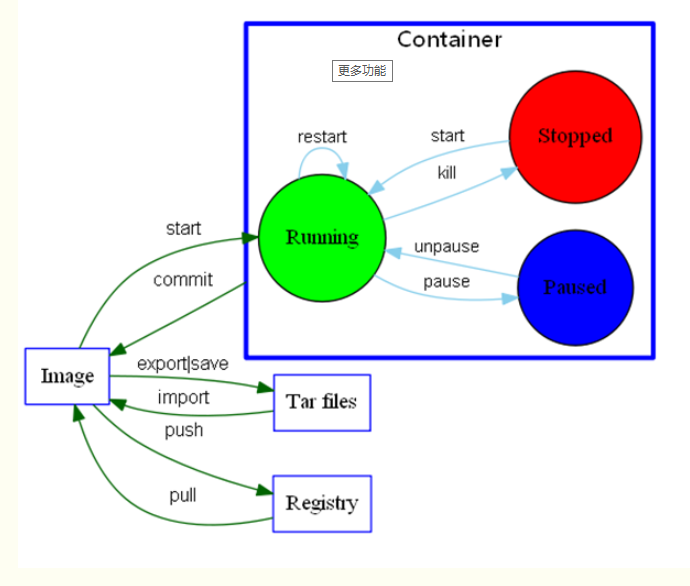
DockerPool：国内专业的docker技术社区，http://www.dockerpool.com也提供官方镜像的下载。

每个仓库中存放着一类镜像，镜像、仓库和registry的关系如下。

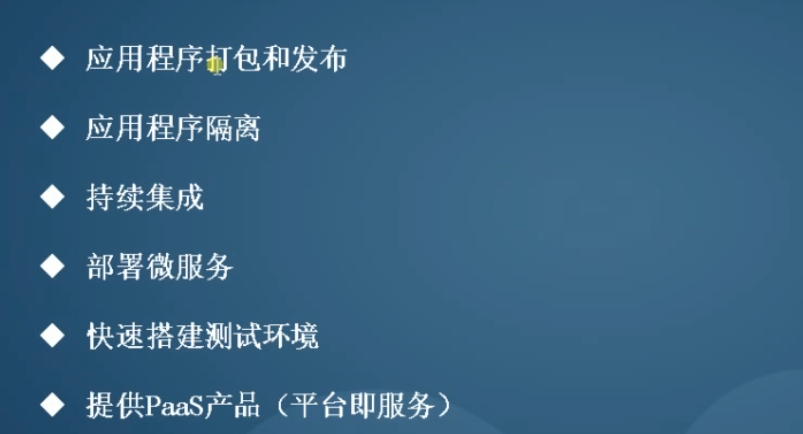


Docker三大核心组件的关系

容器、镜像、仓库的运行关系图



### 5 应用场景



## 二 Docker安装

Docker官方文档:

Docs.docker.com

新版本的额docker支持centos7及以上的linux.

Centos 7的安装方式，请参考官方文档。

安装时，关闭防火墙。

关闭SELINUX（为啥我也不知道。）

生成环境中一般使用17.12版本

### 1 Ubuntu 16.04 安装 Docker

1.选择国内的云服务商，这里选择阿里云为例

curl -sSL http://acs-public-mirror.oss-cn-hangzhou.aliyuncs.com/docker-engine/internet | sh -

2.安装所需要的包

sudo apt-get install linux-image-extra-$(uname -r) linux-image-extra-virtual

3.添加使用 HTTPS 传输的软件包以及 CA 证书

sudo apt-get update

sudo apt-get install apt-transport-https ca-certificates

4.添加GPG密钥

sudo apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80 --recv-keys 58118E89F3A912897C070ADBF76221572C52609D

5.添加软件源

echo "deb https://apt.dockerproject.org/repo ubuntu-xenial main" | sudo tee /etc/apt/sources.list.d/docker.list

6.添加成功后更新软件包缓存

sudo apt-get update

7.安装docker

sudo apt-get install docker-engine

8.设置开机自启动 docker，并启动docker-cer

sudo systemctl enable docker k ----设置docker开机自启动

sudo systemctl start docker ----启动docker

### 2 Ubuntu 18.04 安装 Docker-ce

1.更换国内软件源，推荐中国科技大学的源，稳定速度快（可选）

sudo cp /etc/apt/sources.list /etc/apt/sources.list.bak

sudo sed -i 's/archive.ubuntu.com/mirrors.ustc.edu.cn/g' /etc/apt/sources.list

sudo apt update

2.安装需要的包

sudo apt install apt-transport-https ca-certificates software-properties-common curl

3.添加 GPG 密钥，并添加 Docker-ce 软件源，这里还是以中国科技大学的 Docker-ce 源为例

curl -fsSL https://mirrors.ustc.edu.cn/docker-ce/linux/ubuntu/gpg | sudo apt-key add -

sudo add-apt-repository "deb [arch=amd64] https://mirrors.ustc.edu.cn/docker-ce/linux/ubuntu \

$(lsb\_release -cs) stable"

4.添加成功后更新软件包缓存

sudo apt update

5.安装 Docker-ce

sudo apt install docker-ce

6.设置开机自启动并启动 Docker-ce（安装成功后默认已设置并启动，可忽略）

sudo systemctl enable docker

sudo systemctl start docker

7.测试运行

sudo docker run hello-world

8.添加当前用户到 docker 用户组，可以不用 sudo 运行 docker（可选）

sudo groupadd docker

sudo usermod -aG docker $USER

9.测试添加用户组（可选）

docker run hello-world

### 3 查看docker的安装信息

Docker info

Docker version

## 三 Docker镜像加速

Ubuntu16.04+、Debian8+、CentOS7

对于使用 systemd 的系统，请在 /etc/docker/daemon.json 中写入如下内容（如果文件不存在请新建该文件）（最简单的方案）：

{"registry-mirrors":["https://registry.docker-cn.com"]}

之后重新启动服务：

$ sudo systemctl daemon-reload

$ sudo systemctl restart docker

|  |
| --- |
| 国内较快的镜像原地址  #Docker 官方中国区  https://registry.docker-cn.com  #网易  http://hub-mirror.c.163.com  #ustc  https://docker.mirrors.ustc.edu.cn |

## 四 Docker镜像使用

镜像学习地址：<https://www.cnblogs.com/sparkdev/p/8901728.html#t1>

### 1 获取docker镜像

1 通过docker image pull 命令获取镜像

本地镜像是运行容器的前提，所以在运行容器前我们需要使用 docker image pull 命令从网络上的镜像仓库把镜像拉取到本地。该命令的格式为：

docker image pull [OPTIONS] NAME[:TAG|@DIGEST]

只指定了镜像的名称，默认会选择拉取 latest 标签标记的镜像。比如我们要拉取最新的 ubuntu 镜像：

$ docker image pull Ubuntu

该命令实际上是从在配置文件中配置的仓库地址中拉取最近的ubuntu镜像。（默认是：DockerHub，下载较慢，可以配置国内的下载地址。）

镜像下载到本地后就可运行容器了，比如：

$ docker run --rm ubuntu echo hello docker

2 通过创建Dockerfile文件获取镜像。

将dockerfile文件放在某个文件夹下。

执行命令 docker build –t imageName:tag

### 2 查看镜像信息

docke image ls

docker images

|  |
| --- |
| 输出的信息中包含的内容有：  REPOSITORY：说明镜像来自哪个仓库，比如 ubuntu 或 registry.docker-cn.com/library/ubuntu。  TAG：镜像的标签信息，比如 14.04 或 latest。  IMAGE ID：标识镜像的 ID 号。  CREATED：创建镜像的时间。  SIZE：镜像大小。 |

为本地镜像添加tag

docker image tag

$ docker image tag ubuntu:14.04 oldubuntu

添加本地镜像添加新的tag

### 3搜索镜像

除了直接在 Docker Hub 的官方网站上搜索镜像资源，还可以通 docker search 命令以命令行的方式进行搜索，比如搜索 mysql 镜像：

$ docker search mysql

选择通过docker search 命令搜索的结果时需要谨慎，一般在docker Hub中查询详细的信息。

### 4删除镜像

Docker image rm +(tag/ID)

两者略有不同

使用tag删除镜像

在通过命令docker image rm Ubuntu:10.0 删除镜像时，如果该镜像的ID对应一个镜像tag，则删除最终的镜像文件。如果该镜像ID对应多个tag，则通过tag删除时，并不同删除镜像文件。因为其他tag还在引用这个镜像文件。当删除最后一个ID对应的tag镜像时，该镜像文件才被彻底删除。

使用ID删除镜像

Docker image rm imgID

通过ID删除镜像时，在镜像ID 只对应一个tag时，可以删除镜像。

如果该ID被多个tag引用，则提示删除错误。

### 5 创建镜像

待续。。。。。

### 6 导出和导入镜像

待续。。。。。

### 7 上传镜像

待续。。。。。

## 五 Docker容器使用

Docker容器学习地址：<https://www.cnblogs.com/whych/p/9446032.html>

### 1 创建容器

1 docker run -i -t REPOSITORY:TAG (新建并启动容器:)

-t:让docker分配一个伪终端并绑定到容器的标准输入上

-i: 让容器的标准输入保持打开

若要在后台以守护态（daemonized）形式运行，可加参数-d

Eg:

Docker run –ti Ubuntu:10.04

Docker run 的执行过程

|  |
| --- |
| 在执行docker run来创建并启动容器时，后台运行的标准包括：  检查本地是否存在指定的镜像，不存在就从公有仓库下载  利用镜像创建并启动一个容器  分配一个文件系统，并在只读的镜像层外面挂载一层可读可写层  从宿主机配置的网桥接口中桥接一个虚拟接口到容器  从地址池配置一个ip地址给容器  执行用户指定的应用程序  执行完毕后容器被终止 |

2 docker create –it Ubuntu

使用docker create命令创建的容器处于停止状态，可以使用docker start命令启动它。

### 2开启/停止/重启容器

docker start/stop/restart <containerID>

### 3进入容器

docker attach [container\_id]

连接一个正在运行的container实例（即实例须为start状态，可以多个 窗口同时attach 一个container实例），但当某个窗口因命令阻塞时，其它窗口也无法执行了。

在使用-d参数时，容器启动后会进入后台，用户无法看到容器中的信息。

exec可直接在容器内运行的命令

docker exec container\_ID

### 4退出容器

Exit

Ctrl+P+Q

### 5 查看容器

docker ps –a

查看所有容器

docker ps

查看正在运行的容器。

### 6 删除容器

docker rm <container...> #：删除一个或多个container(删除停止的容器。)

docker rm `docker ps -a -q` #：删除所有的container

docker ps -a -q | xargs docker rm #：同上, 删除所有的container

docker -rm

-f 强制中止并运行的容器

-l 删除容器的连接，但保留容器

-v 删除容器挂载的数据卷

### 7修改容器

docker commit <container> [repo:tag]

将一个container固化为一个新的image，后面的repo:tag可选。

### 8 导出容器

待续。。。。

### 9 查看版本信息

docker –v #简单查看版本  
docker version #查看版本

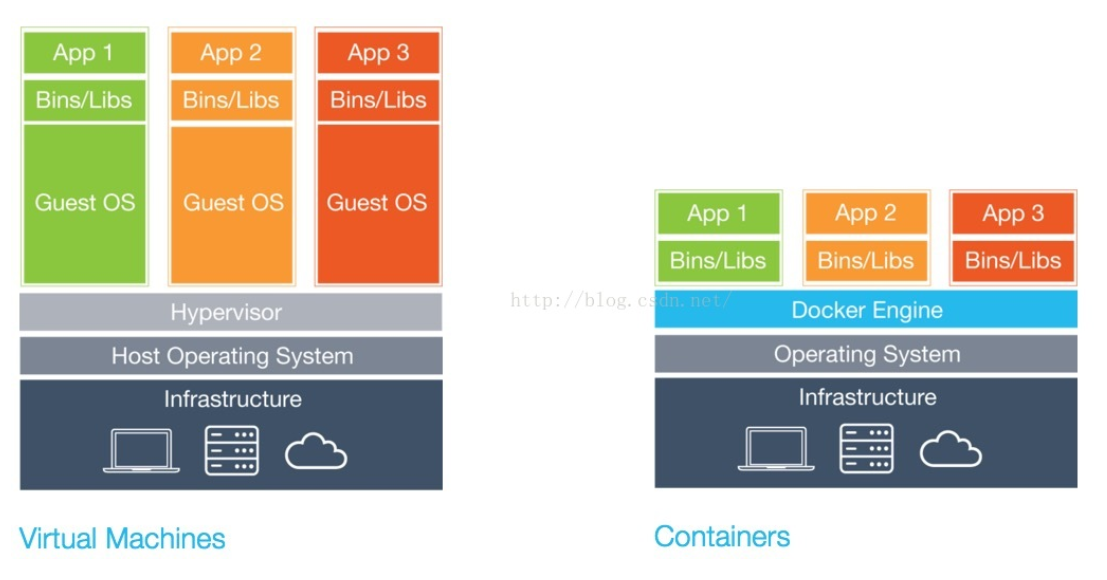
docker info #查看信息

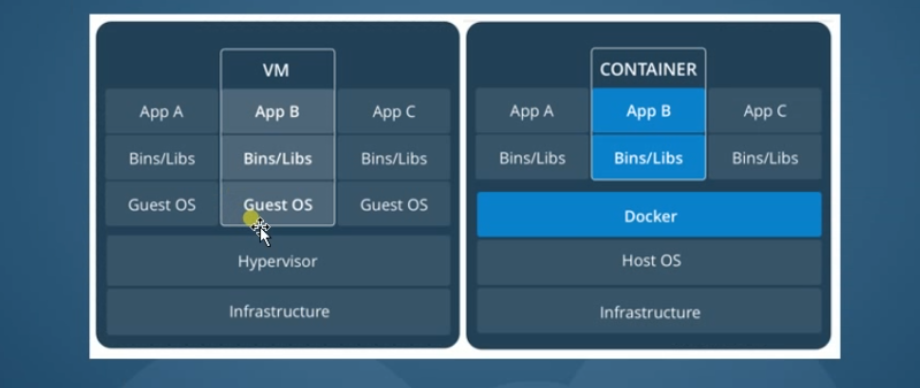
## 六 Docker仓库使用

## 七 Docker卸载

$ sudo apt-get remove docker docker-engine docker.io containerd runc

## 八 Docker与虚拟机的关系和区别



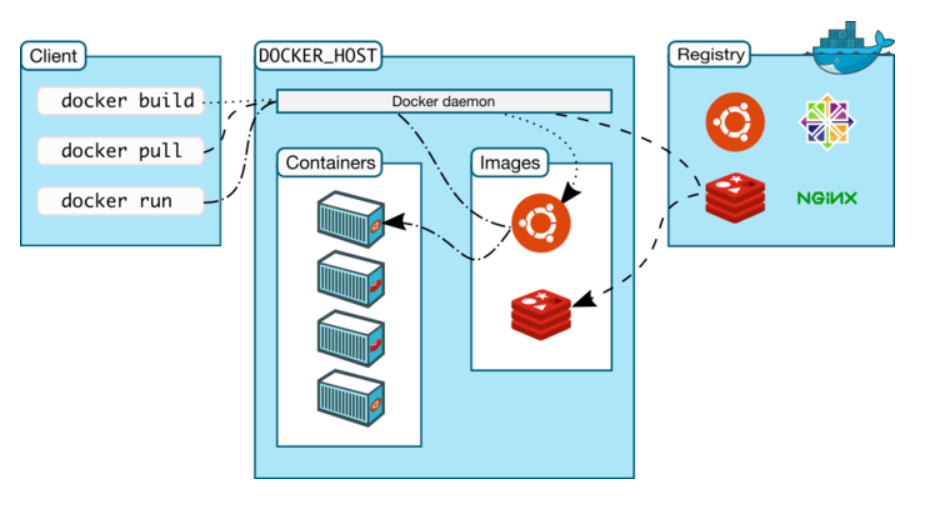


每个虚拟机不仅包括程序、程序需要的包，而且还包含了一个操作系统，但是Docker容器是共享一个操作系统内核（kernel）的，这些容器通过命名空间相互独立。

虚拟机实现资源隔离和环境隔离的方法是利用独立的OS，并利用Hypervisor（Hypervisor是一种运行在物理服务器和操作系统之间的中间软件层,可允许多个操作系统和应用共享一套基础物理硬件。当服务器启动并执行Hypervisor时，它会给每一台虚拟机分配适量的内存、CPU、网络和磁盘，并加载所有虚拟机的客户操作系统。）虚拟化CPU、内存、IO设备等实现的。

对比虚拟机实现资源和环境隔离的方案，Docker就显得简练很多。Docker Engine可以简单看成对Linux的NameSpace、Cgroup、镜像管理文件系统操作的封装。Docker并没有和虚拟机一样利用一个完全独立的Guest OS实现环境隔离，它利用的是目前Linux内核本身支持的容器方式实现资源和环境隔离。简单的说，Docker利用namespace实现系统环境的隔离；利用Cgroup实现资源限制；利用镜像实现根目录环境的隔离。

## 九 Docker运行流程



正如上图所示，当我们想运行一个容器的时候，docker会：

1、拉取镜像，若本地已经存在该镜像，则不用到网上去拉取

2、创建新的容器

3、分配union文件系统并且挂着一个可读写的层，任何修改容器的操作都会被记录在这个读写层上，你可以保存这些修改成新的镜像，也可以选择不保存，那么下次运行改镜像的时候所有修改操作都会被消除

4、分配网络\桥接接口，创建一个允许容器与本地主机通信的网络接口

5、设置ip地址，从池中寻找一个可用的ip地址附加到容器上，换句话说，localhost并不能访问到容器

6、运行你指定的程序

7、捕获并且提供应用输出，包括输入、输出、报错信息

## 十 Docker中安装常用软件

<https://www.jianshu.com/p/17c5811cd850>

### 1 docker中安装mysql

下载镜像

docker pull mysql:5.7

不加tag，默认下载lastst(8.0版本)

不建议安装8.0,新的版本改动了部分内容，装了容易出现问题。

安装8.0的教程：<https://blog.csdn.net/weixin_40693633/article/details/84074459>

创建mysql容器，启动mysql服务器

docker run --name mysql -p 3306:3306 -e MYSQL\_ROOT\_PASSWORD=root -d mysql:5.7

|  |
| --- |
| # mysql给该容器搞一个标识 --name  # 3306:3306映射容器端口到本机端口 -p  # root为数据库密码 -e：设置环境变量  -d 后台运行容器，并返回容器ID  # latest为mysql版本，此处表示最新版本，可以不填写  MYSQL\_ROOT\_PASSWORD=root 指定mysql的密码，用户名默认为root，注意如果没有指定密码，会启动失败 |

进入容器

docker exec -it mysql bash

登陆mysql

mysql -u root –proot

切换database

Use mysql;

查看当前数据库中的表；

show tables;

执行sql

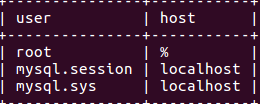
Select user,host from user;

将root用户设置为可远程访问数据库

update user set host ='％'where user ='root';

如果在查询结果中，发现user列有两个root用户，此时删除其中host为localhost的那一列

Delete from user where user=”root” and host=”localhost”;



结果如上图

授权

GRANT ALL PRIVILEGES ON \*.\* TO 'root'@'%' IDENTIFIED BY '你的密码' WITH GRANT OPTION;

flush privileges;

按照上面的设置完成后则可以通过navicat访问mysql。

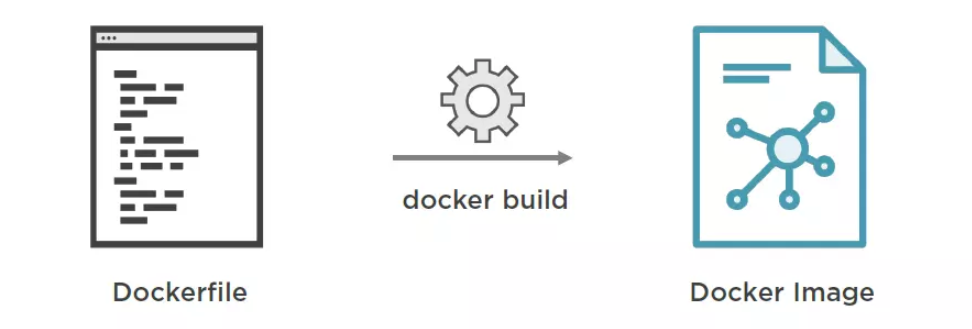
## 十一 Dockerfile

### 1 关于dockerfile

Docker 中用于定义镜像自动化构建流程的配置文件

包含了构建镜像过程中需要执行的命令和其他操作

文件中有注释和指令，dockerfile 中有自己的指令集



### 2 dockerfile的作用

在一个完整的开发、测试、部署过程中，程序运行环境的定义通常是由开发人员来进行的，因为他们更加熟悉程序运转的各个细节，更适合搭建适合程序的运行环境。

在这样的前提下，为了方便测试和运维搭建相同的程序运行环境，常用的做法是由开发人员编写一套环境搭建手册，帮助测试人员和运维人员了解环境搭建的流程。

而 Dockerfile 就很像这样一个环境搭建手册，因为其中包含的就是一个构建容器的过程。

而比环境搭建手册更好的是，Dockerfile 在容器体系下能够完成自动构建，既不需要测试和运维人员深入理解环境中各个软件的具体细节，也不需要人工执行每一个搭建流程。

### 3 dockerfile文档的结构

Dockerfile文档一般从上往下执行，第一行必须是，第一条指令必须是FROM。

Dockerfile 一般分为四部分：基础镜像信息、维护者信息、镜像操作指令和容器启动时执行指令，’#’ 为 Dockerfile 中的注释。

### 4 常用指令

#### FROM

指定基础镜像，必须为第一个命令

|  |
| --- |
| 格式：  　　FROM <image>  　　FROM <image>:<tag>  　　FROM <image>@<digest>  示例：  FROM mysql:5.6  注：  　　tag或digest是可选的，如果不使用这两个值时，会使用latest版本的基础镜像 |

#### MAINTAINER

维护者信息

|  |
| --- |
| 格式：  MAINTAINER <name>  示例：  MAINTAINER Jasper Xu  MAINTAINER sorex@163.com  MAINTAINER Jasper Xu <sorex@163.com> |

#### RUN

构建镜像时执行的命令

|  |
| --- |
| RUN用于在镜像容器中执行命令，其有以下两种命令执行方式：  **shell执行**  格式：  RUN <command>  **exec执行**  格式：  RUN ["executable", "param1", "param2"]  示例：  RUN ["executable", "param1", "param2"]  RUN apk update  RUN ["/etc/execfile", "arg1", "arg1"]  注： 　　RUN指令创建的中间镜像会被缓存，并会在下次构建中使用。如果不想使用这些缓存镜像，可以在构建时指定--no-cache参数，如：docker build --no-cache |

#### 4）ADD

将本地文件添加到容器中，tar类型文件会自动解压(网络压缩资源不会被解压)，可以访问网络资源，类似wget

|  |
| --- |
| 格式：  ADD <src>... <dest>  ADD ["<src>",... "<dest>"] 用于支持包含空格的路径  示例：  ADD hom\* /mydir/ # 添加所有以"hom"开头的文件  ADD hom?.txt /mydir/ # ? 替代一个单字符,例如："home.txt"  ADD test relativeDir/ # 添加 "test" 到 `WORKDIR`/relativeDir/  ADD test /absoluteDir/ # 添加 "test" 到 /absoluteDir/ |

#### 5）COPY

功能类似ADD，但是是不会自动解压文件，也不能访问网络资源

#### 6）CMD

构建容器后调用，也就是在容器启动时才进行调用。

|  |
| --- |
| 格式：  CMD ["executable","param1","param2"] (执行可执行文件，优先)  CMD ["param1","param2"] (设置了ENTRYPOINT，则直接调用ENTRYPOINT添加参数)  CMD command param1 param2 (执行shell内部命令)  示例：  CMD echo "This is a test." | wc -  CMD ["/usr/bin/wc","--help"] 注：  　　CMD不同于RUN，CMD用于指定在容器启动时所要执行的命令，而RUN用于指定镜像构建时所要执行的命令。 |

#### 7）ENTRYPOINT

配置容器，使其可执行化。配合CMD可省去"application"，只使用参数。

|  |
| --- |
| 格式：  ENTRYPOINT ["executable", "param1", "param2"] (可执行文件, 优先)  ENTRYPOINT command param1 param2 (shell内部命令)  示例：  FROM ubuntu  ENTRYPOINT ["top", "-b"]  CMD ["-c"] 注： 　　　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。 |

#### 8）LABEL

用于为镜像添加元数据

|  |
| --- |
| 格式：  LABEL <key>=<value> <key>=<value> <key>=<value> ...  示例：  　　LABEL version="1.0" description="这是一个Web服务器" by="IT笔录"  注：  　　使用LABEL指定元数据时，一条LABEL指定可以指定一或多条元数据，指定多条元数据时不同元数据之间通过空格分隔。推荐将所有的元数据通过一条LABEL指令指定，以免生成过多的中间镜像。 |

#### 9）ENV

设置环境变量

|  |
| --- |
| 格式：  ENV <key> <value> #<key>之后的所有内容均会被视为其<value>的组成部分，因此，一次只能设置一个变量  ENV <key>=<value> ... #可以设置多个变量，每个变量为一个"<key>=<value>"的键值对，如果<key>中包含空格，可以使用\来进行转义，也可以通过""来进行标示；另外，反斜线也可以用于续行  示例：  ENV myName John Doe  ENV myDog Rex The Dog  ENV myCat=fluffy |

#### 10）EXPOSE

指定于外界交互的端口

|  |
| --- |
| 格式：  EXPOSE <port> [<port>...]  示例：  EXPOSE 80 443  EXPOSE 8080  EXPOSE 11211/tcp 11211/udp  注：  　　EXPOSE并不会让容器的端口访问到主机。要使其可访问，需要在docker run运行容器时通过-p来发布这些端口，或通过-P参数来发布EXPOSE导出的所有端口 |

#### 11）VOLUME

用于指定持久化目录

|  |
| --- |
| 格式：  VOLUME ["/path/to/dir"]  示例：  VOLUME ["/data"]  VOLUME ["/var/www", "/var/log/apache2", "/etc/apache2"  注：  　　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：  1 卷可以容器间共享和重用  2 容器并不一定要和其它容器共享卷  3 修改卷后会立即生效  4 对卷的修改不会对镜像产生影响  5 卷会一直存在，直到没有任何容器在使用它 |

#### 12）WORKDIR

工作目录，类似于cd命令

|  |
| --- |
| 格式：  WORKDIR /path/to/workdir  示例：  WORKDIR /a (这时工作目录为/a)  WORKDIR b (这时工作目录为/a/b)  WORKDIR c (这时工作目录为/a/b/c)  注：  　　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。在使用docker run运行容器时，可以通过-w参数覆盖构建时所设置的工作目录。 |

#### 13）USER

指定运行容器时的用户名或 UID，后续的 RUN 也会使用指定用户。使用USER指定用户时，可以使用用户名、UID或GID，或是两者的组合。当服务不需要管理员权限时，可以通过该命令指定运行用户。并且可以在之前创建所需要的用户

|  |
| --- |
| 格式:  　　USER user  　　USER user:group  　　USER uid  　　USER uid:gid  　　USER user:gid  　　USER uid:group  示例：  　　USER www  注：  　　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。 |

#### 14）ARG

用于指定传递给构建运行时的变量

|  |
| --- |
| 格式：  ARG <name>[=<default value>]  示例：  ARG site  ARG build\_user=www |

#### 15）ONBUILD

用于设置镜像触发器

|  |
| --- |
| 格式：  　　ONBUILD [INSTRUCTION]  示例：  　　ONBUILD ADD . /app/src  　　ONBUILD RUN /usr/local/bin/python-build --dir /app/src  注：  　　当所构建的镜像被用做其它镜像的基础镜像，该镜像中的触发器将会被钥触发 |

### 5 图解



### 6 实例

|  |
| --- |
| # This my first nginx Dockerfile  # Version 1.0  # Base images 基础镜像  FROM centos  #MAINTAINER 维护者信息  MAINTAINER tianfeiyu  #ENV 设置环境变量  ENV PATH /usr/local/nginx/sbin:$PATH  #ADD 文件放在当前目录下，拷过去会自动解压  ADD nginx-1.8.0.tar.gz /usr/local/  ADD epel-release-latest-7.noarch.rpm /usr/local/  #RUN 执行以下命令  RUN rpm -ivh /usr/local/epel-release-latest-7.noarch.rpm  RUN yum install -y wget lftp gcc gcc-c++ make openssl-devel pcre-devel pcre && yum clean all  RUN useradd -s /sbin/nologin -M www  #WORKDIR 相当于cd  WORKDIR /usr/local/nginx-1.8.0  RUN ./configure --prefix=/usr/local/nginx --user=www --group=www --with-http\_ssl\_module --with-pcre && make && make install  RUN echo "daemon off;" >> /etc/nginx.conf  #EXPOSE 映射端口  EXPOSE 80  #CMD 运行以下命令  CMD ["nginx"] |

## 十二 搭建私有镜像库

## =========================

## 问题整理

### 1 Ubuntu下docker使用非root权限运行docker

默认情况下，docker 命令会使用 Unix socket 与 Docker 引擎通讯。而只有 root 用户和 docker 组的用户才可以访问 Docker 引擎的 Unix socket。出于安全考虑，一般 Linux 系统上不会直接使用 root 用户。因此，更好地做法是将需要使用 docker 的用户加入 docker 用户组。

1.添加 docker group ：

$ sudo groupadd docker

2.将用户加入该 group 内

$ sudo usermod -aG docker $USER

# 或者使用下面命令

$ sudo gpasswd -a ${USER} docker

3.重启服务

$ sudo service docker restart

# 或者

$ sudo /etc/init.d/docker restart

4.切换当前会话到新 group 或者重启 X 会话

注意:这一步是必须的，否则因为 groups 命令获取到的是缓存的组信息，刚添加的组信息未能生效，所以 docker images 执行时同样有错。

$ newgrp – docker

### 2 Docker启动/重启/停止

启动       systemctl start docker

守护进程重启   sudo systemctl daemon-reload

重启docker服务   systemctl restart  docker

重启docker服务  sudo service docker restart

关闭docker service docker stop

关闭docker systemctl stop docker