Docker 是一个[开源](https://baike.baidu.com/item/%E5%BC%80%E6%BA%90/246339)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容器中，然后发布到任何流行的[Linux](https://baike.baidu.com/item/Linux)机器上，也可以实现[虚拟化](https://baike.baidu.com/item/%E8%99%9A%E6%8B%9F%E5%8C%96/547949)。

一个完整的Docker有以下几个部分组成：

Docker Client客户端

Docker Daemon守护进程

Docker Image镜像

Docker Container容器

Docker采用 C/S架构 Docker daemon 作为服务端接受来自客户的请求，并处理这些请求（创建、运行、分发容器）。 客户端和服务端既可以运行在一个机器上，也可通过 socket 或者RESTful API 来进行通信。

# 一、基本概念：

**镜像(Image)**

镜像，从认识上简单的来说，就是面向对象中的类，相当于一个模板。从本质上来说，镜像相当于一个文件系统。Docker 镜像是一个特殊的文件系统，除了提供容器运行时所需的程序、库、资源、配置等文件外，还包含了一些为运行时准备的一些配置参数（如匿名卷、环境变量、用户等）。镜像不包含任何动态数据，其内容在构建之后也不会被改变。

**容器(Container)**

容器，从认识上来说，就是类创建的实例，就是依据镜像这个模板创建出来的实体。容器的实质是进程，但与直接在宿主执行的进程不同，容器进程运行于属于自己的独立的命名空间。因此容器可以拥有自己的root 文件系统、自己的网络配置、自己的进程空间，甚至自己的用户ID 空间。容器内的进程是运行在一个隔离的环境里，使用起来，就好像是在一个独立于宿主的系统下操作一样。这种特性使得容器封装的应用比直接在宿主运行更加安全。

**仓库(Repository)**

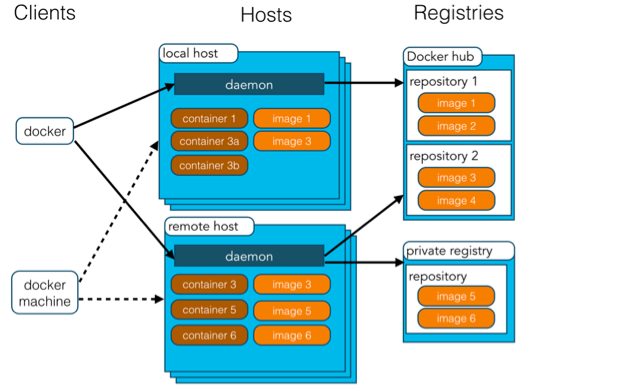
仓库，从认识上来说，就好像软件包上传下载站，有各种软件的不同版本被上传供用户下载。镜像构建完成后，可以很容易的在当前宿主机上运行，但是，如果需要在其它服务器上使用这个镜像，我们就需要一个集中的存储、分发镜像的服务，Docker Registry 就是这样的服务。

**分层存储**

因为镜像包含操作系统完整的root 文件系统，其体积往往是庞大的，因此在Docker设计时，就充分利用Union FS 的技术，将其设计为分层存储的架构。所以严格来说，镜像并非是像一个ISO 那样的打包文件，镜像只是一个虚拟的概念，其实际体现并非由一个文件组成，而是由一组文件系统组成，或者说，由多层文件系统联合组成。

镜像构建时，会一层层构建，前一层是后一层的基础。每一层构建完就不会再发生改变，后一层上的任何改变只发生在自己这一层。比如，删除前一层文件的操作，实际不是真的删除前一层的文件，而是仅在当前层标记为该文件已删除。在最终容器运行的时候，虽然不会看到这个文件，但是实际上该文件会一直跟随镜像。因此，在构建镜像的时候，需要额外小心，每一层尽量只包含该层需要添加的东西，任何额外的东西应该在该层构建结束前清理掉。

**Docker 架构**



|  |  |
| --- | --- |
| 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](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。 |

# 二、Docker安装

Centos7 下安装

安装一些系统工具：sudo yum install -y yum-utils device-mapper-persistent-data lvm2

添加软件源信息：sudo yum-config-manager --add-repo <http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo>

更新yum： sudo yum makecache fast

安装docker-ce：sudo yum install docker-ce

启动服务 sudo systemctl start docker

测试：docker run hello-world

**更换镜像：否则会下载速度较慢，***/etc/docker/***daemon.json文件添加**

{

"registry-mirrors": ["http://hub-mirror.c.163.com"]

}

# 三、Docker 镜像使用

**使用 docker images 来列出本地主机上的镜像**

runoob@runoob:~$ docker images

创建镜像

当我们从docker镜像仓库中下载的镜像不能满足我们的需求时，我们可以通过以下两种方式对镜像进行更改。

1.从已经创建的容器中更新镜像，并且提交这个镜像

2.使用 Dockerfile 指令来创建一个新的镜像

**使用 docker pull 命令来下载想预先下载的镜像**

runoob@runoob:~$ docker pull ubuntu:13.10

**通过容器构建镜像：*docker commit [OPTIONS] CONTAINER [REPOSITORY[:TAG]]***

**通过dockerfile：docker build**利用一个类似配置文件的形式来进行构建自定义镜像

运行一个web 应用

docker run -d -P training/webapp python app.py

docker run -d -p 5000:5000 training/webapp python app.py

参数说明:

* **-d:**让容器在后台运行。
* **-P:**将容器内部使用的网络端口映射到我们使用的主机上（随机）
* **-**p自定义容器内部的 5000 端口映射到我们本地主机的 5000 端口上（自定义）

使用 **docker port** 可以查看指定 （ID 或者名字）容器的某个确定端口映射到宿主机的端口号

docker tag 镜像ID 名称：标签 为镜像添加一个标签

# 四、容器操作（常见命令）

**运行交互式的容器**：runoob@runoob:~$ docker run -i -t ubuntu:15.10 /bin/bash

运行命令**cat /proc/version**和**ls**分别查看当前系统的版本信息和当前目录下的文件列表

**启动容器（后台模式）**：runoob@runoob:~$ docker run -d ubuntu:15.10 /bin/sh -c "while true; do echo hello world; sleep 1; done"

（输出为容器id）

**查看运行的docker信息：** runoob@runoob:~$ docker ps

**查看所有的docker信息：** runoob@runoob:~$ docker ps -a

**查看容器的标准输出：** runoob@runoob:~$ docker logs 容器id或者容器名称

**查看Docker客户端的所有命令选项：r**unoob@runoob:~# docker

**删除容器：** docker rm 容器名

**停止容器：** docker stop 容器名

**重新启动：**  docker restart 容器名

# 五、docker如何push镜像到docker hub 个人仓库：

1 找到本地镜像的ID，docker images

2 登录docker hub，docker login

3 定义新的tag： docker tag 镜像id 用户名/镜像名：version tag latest（版本号）

4 push镜像： docker push 用户名/镜像名

备注：用户名/镜像名 只是一种命名规范，可以不遵守；如果在docker hub已经构建个人仓库，此处应该以个人仓库名（用户名/仓库名）为准，因为默认免费构建1个个人仓库，其他的虽然显示push成功但无法上传至个人仓库。直接push不在docker hub上手动建个人仓库也可以实现。一个仓库可以存多个镜像。

涉及到镜像删除时，有些镜像的ID相同导致存在于不同的仓库，这就需要首先删除指定的仓库及标签：即 docker rmi repository：tag，然后docker rmi imageid

# 六、Dockerfile指令详解：

1 FROM 指定基础镜像 FROM image：tag

通过 FROM 指定的镜像，可以是任何有效的基础镜像。

FROM 有以下限制：

FROM 必须是 Dockerfile 中第一条非注释命令;

在一个 Dockerfile 文件中创建多个镜像时，FROM 可以多次出现。只需在每个新命令 FROM 之前，记录提交上次的镜像 ID;

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

**2 RUN 执行命令**

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。

#shell格式 RUN <command>

#exec格式 RUN ["executable", "param1", "param2"]

* RUN 命令将在当前 image 中执行任意合法命令并提交执行结果。命令执行提交后，就会自动执行 Dockerfile 中的下一个指令。
* 层级 RUN 指令和生成提交是符合 Docker 核心理念的做法。它允许像版本控制那样，在任意一个点，对 image 镜像进行定制化构建。
* 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

**3 COPY 复制文件**

COPY <源路径>... <目标路径>

COPY ["<源路径1>",... "<目标路径>"]

目标路径不需要事先创建，如果目录不存在会在复制文件前先行创建缺失目录。

**4 ADD 更高级的复制文件**

**5 ENV 设置环境变量**

ENV <key> <value>

ENV <key1>=<value1> <key2>=<value2>...

**6 EXPOSE**

为构建的镜像设置监听端口，使容器在运行时监听。

EXPOSE <port> [<port>...]

**7 VOLUME 定义匿名卷**

VOLUME用于创建挂载点，即向基于所构建镜像创始的容器添加卷：

VOLUME ["/data"]

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

8 WORKDIR 指定工作目录

WORKDIR /path/to/workdir

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

9 USER 指定当前用户

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

10 CMD

CMD用于指定在容器启动时所要执行的命令。CMD 有以下三种格式：

CMD ["executable","param1","param2"]

CMD ["param1","param2"]

CMD command param1 param2

与 RUN 指令的区别：RUN 在构建的时候执行，并生成一个新的镜像，CMD 在容器运行的时候执行，在构建时不进行任何操作。

11 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：

12 LABEL

LABEL用于为镜像添加元数据，元数以键值对的形式指定：

13 ARG

ARG用于指定传递给构建运行时的变量：

14 ONBUILD

ONBUILD用于设置镜像触发器：

15 STOPSIGNAL

STOPSIGNAL用于设置停止容器所要发送的系统调用信号：

16 SHELL

SHELL用于设置执行命令（shell式）所使用的的默认 shell 类型：

# 七、Docker compose指令详解

Compose 是 Docker 容器进行编排的工具，定义和运行多容器的应用，可以一条命令启动多个容器，使用Docker Compose不再需要使用shell脚本来启动容器。

Compose 通过一个配置文件来管理多个Docker容器，在配置文件中，所有的容器通过services来定义，然后使用docker-compose脚本来启动，停止和重启应用，和应用中的服务以及所有依赖服务的容器，非常适合组合使用多个容器进行开发的场景。

docker-compose默认的模板文件是 docker-compose.yml，其中定义的每个服务都必须通过 image 指令指定镜像或 build 指令（需要 Dockerfile）来自动构建。如果使用 build 指令，在 Dockerfile 中设置的选项(例如：CMD, EXPOSE, VOLUME, ENV 等) 将会自动被获取，无需在 docker-compose.yml 中再次设置。

使用Compose 基本上分为三步：**1.Dockerfile 定义应用的运行环境；2.docker-compose.yml 定义组成应用的各服务；3.docker-compose up 启动整个应用**

标准配置文件应该包含 version、services、networks 三大部分，其中最关键的就是 services 和 networks 两个部分。compose 文件是一个定义服务、网络和卷的 YAML 文件。

# 八、实例

1 较为简单的环境：

1.1 构建python环境运行tests.py

第一步：docker pull python：3.7

第二步：docker run -p 5000:5000 –v /data/flask\_demo: /data/workdir -w /data/workdir python：3.7 python tests.py

-v 将主机文件目录映射到运行目录

-w 运行目录

python：3.7 镜像名

1.2构建python环境运行tests.py

将tests.py代码融入Dockerfile构建镜像的语句中内，从而生成容器后即可出现结果。

第一步：vi Dockerfile

第二步：FROM python：3.7

RUN (可以添加很多依赖环境)

COPY . /data/flask\_demo

WORKDIR /data/flask\_demo

ENTRYPOINT [“python”]

CMD [“./ tests.py”]

第三步： docker bulid 镜像名：tag . （.表示Dockerfile 文件所在目录，可以指定Dockerfile 的绝对路径）

2 复杂的环境，依赖多个环境：

Dockerfile 可以让用户管理一个单独的应用容器；而 Compose 则允许用户在一个模板（YAML 格式）中定义一组相关联的应用容器（被称为一个 project，即项目）

2.1 运用docker-compose构建python,flask环境运行run.py

第一步: 构建一个项目文件夹 data

第二步：运行的代码在此文件夹下面

第三步：创建 requirements.txt 添加项目依赖的python包：（flask，scrapy, scrapyd，测试文件仅涉及flask）

第四步：创建Dockerfile，

FROM python:3.7

ADD . /code

WORKDIR /code

RUN pip install -r requirements.txt

CMD ["python", "run.py"]

第五步：创建 docker-compose.yml文件

version: '2'

services:

web:

build: .

ports:

- "8000:5000"

第六步：docker-compose up （运行容器）

docker-compose bulid （创建服务容器）

调用镜像生成容器 docker run -d -p 5000:5000 -v /data/flask\_demo:/data -w /data flaskdemo\_web python run.py

2.2 用Dockerfile

使用Dockerfile搭建jdk、tomcat运行环境

第一步：创建dockerfile

vi Dockerfile

# 版本信息

FROM centos

MAINTAINER locutus "locutus@foxmail.com"

# OS环境配置

RUN yum install -y wget

# 安装JDK

RUN mkdir /var/tmp/jdk

RUN wget --no-check-certificate --no-cookies --header "Cookie: oraclelicense=accept-securebackup-cookie" -P /var/tmp/jdk http://download.oracle.com/otn-pub/java/jdk/8u111-b14/jdk-8u111-linux-x64.tar.gz

RUN tar xzf /var/tmp/jdk/jdk-8u111-linux-x64.tar.gz -C /var/tmp/jdk && rm -rf /var/tmp/jdk/jdk-8u111-linux-x64.tar.gz

# 安装tomcat

RUN mkdir /var/tmp/tomcat

RUN wget -P /var/tmp/tomcat http://archive.apache.org/dist/tomcat/tomcat-8/v8.5.8/bin/apache-tomcat-8.5.8.tar.gz

RUN tar xzf /var/tmp/tomcat/apache-tomcat-8.5.8.tar.gz -C /var/tmp/tomcat && rm -rf /var/tmp/tomcat/apache-tomcat-8.5.8.tar.gz

#设置环境变量

ENV JAVA\_HOME /var/tmp/jdk/jdk1.8.0\_111

ENV CATALINA\_HOME /var/tmp/tomcat/apache-tomcat-8.5.8

ENV PATH $PATH:$JAVA\_HOME/bin:$CATALINA\_HOME/bin

#打包项目并拷贝到tomcat webapps目录

RUN mkdir /var/tmp/webapp

ADD ./ /var/tmp/webapp

RUN cd /var/tmp/webapp && cp /var/tmp/webapp/war/sm\_new.war /var/tmp/tomcat/apache-tomcat-8.5.8/webapps/

#开启内部服务端口

EXPOSE 8080

#启动tomcat服务器

CMD ["./var/tmp/tomcat/apache-tomcat-8.5.8/bin/catalina.sh","run"] && tail -f /var/tmp/tomcat/apache-tomcat-8.5.8/logs/catalina.out

第二步：生成镜像： docker build -t tomcat .

**总结：**

**从仓库（一般为Docker Hub）下载（pull）一个镜像，Docker执行run方法得到一个容器，用户在容器里执行各种操作。Docker执行commit方法将一个容器转化为镜像。Docker利用login、push等命令将本地镜像推送（push）到仓库。其他机器或服务器上就可以使用该镜像去生成容器，进而运行相应的应用程序了。**

# 九 常见错误

错误1：yum安装软件出现 Cannot find a valid baseurl for repo: base/7/x86\_64 #可能是网络问题，查看resolv.conf发现无内容

解决： 更改 /etc/resolv.conf，加入

nameserver 8.8.8.8

nameserver 4.2.2.2

nameserver 172.19.0.6

nameserver 172.19.0.5

错误2：No package docker-ce available.

解决： sudo yum install epel-release

错误3： Unable to find image 'hellow-world:latest' locally 无法从国外网站获取镜像

解决： 修改docker文件 sudo vi /etc/default/docker

DOCKER\_OPTS="--registry-mirror=http://yourRegist.m.daocloud.io"

service docker restart

错误4：无法注册docker hub账户

解决：访问https://hideuvpn.gitlab.io，注册新用户，获取代理，访问www.hub.docker.com官网,下方出现人脸识别说明成功

错误5： docker: invalid reference format.

解决： 注意“-”的书写方式