Docker

作为一种新兴的虚拟化方式，Docker 跟传统的虚拟化方式相比具有众多的优势。

（1）更高效的利用系统资源。

由于容器不需要进行硬件虚拟以及运行完整操作系统等额外开销，Docker 对系统资源的利用率更高。无论是应用执行速度、内存损耗或者文件存储速度，都要比传统虚拟机技术更高效。因此，相比虚拟机技术，一个相同配置的主机，往往可以运行更多数量的应用。

（2）更快速的启动时间

传统的虚拟机技术启动应用服务往往需要数分钟，而 Docker 容器应用，由于直接运行于宿主内核，无需启动完整的操作系统，因此可以做到秒级、甚至毫秒级的启动时间。大大的节约了开发、测试、部署的时间。

（3）一致的运行环境

开发过程中一个常见的问题是环境一致性问题。由于开发环境、测试环境、生产环境不一致，导致有些 bug 并未在开发过程中被发现。而 Docker 的镜像提供了除内核外完整的运行时环境，确保了应用运行环境一致性，从而不会再出现 「这段代码在我机器上没问题啊」 这类问题。

（4）持续交付和部署

对开发和运维（DevOps）人员来说，最希望的就是一次创建或配置，可以在任意地方正常运行。

使用 Docker 可以通过定制应用镜像来实现持续集成、持续交付、部署。开发人员可以通过 Dockerfile 来进行镜像构建，并结合 持续集成(Continuous Integration) 系统进行集成测试，而运维人员则可以直接在生产环境中快速部署该镜像，甚至结合 持续部署(Continuous Delivery/Deployment) 系统进行自动部署。

而且使用 Dockerfile 使镜像构建透明化，不仅仅开发团队可以理解应用运行环境，也方便运维团队理解应用运行所需条件，帮助更好的生产环境中部署该镜像。

（5）更轻松的迁移

由于 Docker 确保了执行环境的一致性，使得应用的迁移更加容易。Docker 可以在很多平台上运行，无论是物理机、虚拟机、公有云、私有云，甚至是笔记本，其运行结果是一致的。因此用户可以很轻易的将在一个平台上运行的应用，迁移到另一个平台上，而不用担心运行环境的变化导致应用无法正常运行的情况。

（6）更轻松的维护和扩展

Docker 使用的分层存储以及镜像的技术，使得应用重复部分的复用更为容易，也使得应用的维护更新更加简单，基于基础镜像进一步扩展镜像也变得非常简单。此外，Docker 团队同各个开源项目团队一起维护了一大批高质量的 官方镜像，既可以直接在生产环境使用，又可以作为基础进一步定制，大大的降低了应用服务的镜像制作成本。

Docker 使用方法

(一) 镜像管理

从Docker Hub查找镜像：

(也可以直接去官方仓库查找，这样信息更直观完整)

docker search [OPTIONS] TERM

# OPTIONS:

# -f,--filter <filter>：列出收藏数不小于指定值的镜像

# filter包括stars=100（收藏数不少于100的镜像），is-official=true（指定Docker官方发布的镜像）

# --limit <num>：设置最多输出num条记录，默认25条

# --no-trunc：不截断，显示完整的镜像描述

eg：

# 从Docker Hub搜索收藏数多于100的官方java镜像

docker search -f stars=100 is-official=true java

从镜像仓库拉取镜像：(拉取的镜像元信息存储在 /var/lib/docker/ 目录下)

docker pull [OPTIONS] NAME[:TAG]

# OPTIONS：

# -a：拉取所有TAG镜像

# --disable-content-trust：忽略镜像的校验，默认开启

eg:

# 从Docker Hub下载java最新版镜像

docker pull java

列出本地镜像：

docker images [OPTIONS] [REPOSITORY[:TAG]]

# OPTIONS：

# -a：列出本地所有的镜像，包括中间映像层

# -q,--quiet：只显示镜像ID

# --digests：显示镜像的摘要信息

# --no-trunc：不截断，显示完整的镜像信息

删除本地镜像：

docker rmi [OPTIONS] IMAGE [IMAGE...]

# OPTIONS：

# -f,--force：强制删除

标记本地镜像，将其归入某一仓库：

docker tag IMAGE[:TAG] [REGISTRYHOST/][USERNAME/]NAME[:TAG]

eg:

# 将镜像ubuntu:latest标记为captain\_rb/ubuntu:v3镜像

docker tag ubuntu:latest captain\_rb/ubuntu:v3

从本地容器创建一个新的镜像：

docker commit [OPTIONS] CONTAINER [REPOSITORY[:TAG]]

# OPTIONS:

# -a,--author <author>：提交的镜像作者

# -c,--change <list>：使用Dockerfile指令来创建镜像

# -m,--message <string>：提交时的说明信息

# -p,--pause：在commit时将容器暂停

将一个或多个镜像打包：

docker save [OPTIONS] IMAGE [IMAGE...]

# OPTIONS:

# -o,--output <atchive>：将镜像打包为tar包

从tar包加载镜像：(可用于离线安装)

docker load [OPTIONS]

# OPTIONS:

# -i,--input <atchive>：从tar包加载镜像

# -q,--quiet：压缩输出信息

使用Dockerfile创建镜像：

docker build [OPTIONS] PATH | URL

# OPTIONS:

# -f,--file <path>：指定要使用的Dockerfile路径

# -t,--tag <list>：镜像名字及标签，通常为 name:tag 或者 name 格式

# -q,--quiet：安静模式，成功后只输出镜像 ID

# --rm：设置镜像成功后删除中间容器

eg:

# 在当前目录下构建名为captain\_rb/myapp:v1的镜像

docker build -t captain\_rb/myapp:v1 .

登入/登出Docker镜像仓库，登入镜像仓库后，才可以拉取/推送私人仓库镜像：

docker login [OPTIONS] [SERVER]

docker logout [OPTIONS] [SERVER]

# OPTIONS：

# -u：登录的用户名

# -p：登录的密码

# SERVER未指定则为官方仓库Docker Hub

eg：

# 登陆前需要在阿里云容器镜像服务->实例列表->访问凭证中设置密码

docker login --username=captain\_rb registry.cn-hangzhou.aliyuncs.com

将本地的镜像上传到镜像仓库，需要先登录：

docker push [OPTIONS] NAME[:TAG]

# OPTIONS：

# --disable-content-trust：忽略镜像的校验，默认开启

(二) 容器管理

创建一个新的容器并启动，运行命令：

docker run [OPTIONS] IMAGE [COMMAND] [ARG...]

# OPTIONS:

# -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

# -t：为容器重新分配一个伪输入终端

# -p：指定端口映射，格式为："主机端口:容器端口"

# -P：随机端口映射，容器内部端口随机映射到主机的端口

# -v，--volume：指定卷的映射，格式为："主机卷:容器卷"

# --name="new name"：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

eg：

docker run -it nginx:latest /bin/bash

docker run -p 80:80 -v /data:/data -d nginx:latest

创建一个新的容器，不启动：

docker create [OPTIONS] IMAGE [COMMAND] [ARG...]

# OPTIONS 同上

在运行的容器中执行命令：

docker exec [OPTIONS] CONTAINER COMMAND [ARG...]

# OPTIONS：

# -d：后台运行

# -i：以交互模式执行命令，通常与-t同时使用

# -t：分配一个伪终端

eg：

# 进入正在运行的nginx容器

docker exec -it nginx /bin/bash

启动/停止/重启 容器：

docker start CONTAINER [CONTAINER...]

docker stop CONTAINER [CONTAINER...]

docker restart CONTAINER [CONTAINER...]

获取容器/镜像的元数据：

docker inspect NAME|ID [NAME|ID...]

列出本地容器：

docker ps [OPTIONS]

# OPTIONS：

# -a：显示所有的容器，包括未运行的

# -f：根据条件过滤显示的内容

# -l：显示最近创建的容器

# -n：列出最近创建的n个容器

# -q：静默模式，只显示容器ID

# -s：显示总的文件大小

# --no-trunc：不截断，输出容器详细信息

杀掉运行中的容器：

docker kill [OPTIONS] CONTAINER [CONTAINER...]

# OPTIONS:

# -s,--signal：向容器发送一个信号

删除容器：

docker rm [OPTIONS] CONTAINER [CONTAINER...]

# OPTIONS:

# -f：通过 SIGKILL 信号强制删除一个运行中的容器

# -l：移除容器间的网络连接，而非容器本身

# -v：删除与容器关联的卷

暂停/恢复容器运行：

docker pause CONTAINER [CONTAINER...]

docker unpause CONTAINER [CONTAINER...]

查看容器中运行的进程信息：

docker top [OPTIONS] CONTAINER

1

用于容器与主机之间的数据拷贝：

docker cp [OPTIONS] CONTAINER:SRC\_PATH DEST\_PATH

docker cp [OPTIONS] SRC\_PATH CONTAINER:DEST\_PATH

# OPTIONS:

# -a,--archive：文档模式，拷贝所有的uid/gid信息

# -L,--follow-link：保持源目标中的链接不变

注意：使用IMAGE ID / CONTAINER ID指定镜像 / 容器，可以通过截断的ID号进行指定，如：

docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

nginx latest 62d49f9bab67 4 days ago 133MB

ubuntu latest 26b77e58432b 2 weeks ago 72.9MB

# 删除镜像nginx:latest

docker rmi 62

(三) Dockerfile

Dockerfile是用于构建镜像的文本文件，文件内容主要包含了基础镜像信息、开发者信息、镜像构建运行指令和容器启动运行指令，指令每执行一次都会在现有镜像基础上新建一层，为避免文件层过多导致镜像臃肿过大，命令应尽量精简。

如以下命令会创建3层镜像：

FROM centos

RUN yum install wget

RUN wget -O redis.tar.gz "http://download.redis.io/releases/redis-5.0.3.tar.gz"

RUN tar -xvf redis.tar.gz

应该改为一句RUN指令，这样只创建1层镜像：

FROM centos

RUN yum install wget \

&& wget -O redis.tar.gz "http://download.redis.io/releases/redis-5.0.3.tar.gz" \

&& tar -xvf redis.tar.gz

常用Dockerfile命令如下所示：

FROM [--platform=<platform>] <image>[:<tag>]

设置镜像使用的基础镜像

RUN <command>（shell格式）

RUN ["executable", "param1", "param2"]（exec格式，推荐使用）

设置镜像构建 (docker build) 时运行的命令

CMD <command>（shell格式）

CMD ["executable","param1","param2"]（exec格式，推荐使用）

CMD ["param1","param2"]（为ENTRYPOINT指令提供参数）

设置容器启动 (docker run) 时运行的命令，如果存在多个CMD指令，仅最后一个生效

ENTRYPOINT ["<executeable>","<param1>","<param2>",...]

设置容器启动 (docker run) 时运行的命令，如果存在多个ENTRYPOINT指令，仅最后一个生效

LABEL <key>=<value>[ = …]

将元数据添加到镜像中

MAINTAINER <author>

设置镜像作者

ENV <key> <value>

ENV <key1>=<value1> [<key2>=<value2>…]

设置环境变量

USER <user>[:<group>]

USER <UID>[:<GID>]

在它之后的RUN、CMD以及ENTRYPOINT指令都会以设置的USER身份执行

EXPOESE <port1>[/<protocol1>] [<port2>[/<protocol2>]…]

指定容器监听的端口

COPY [–chown=<user>:<group>] "<src1>",... "<dest>"

COPY [–chown=<user>:<group>] <src1>... <dest>

从上下文路径复制文件到容器里指定路径，支持通配符表达式，–chown为可选参数，改变复制到容器内文件的拥有者和属组

ADD [–chown=<user>:<group>] "<src1>",... "<dest>"

ADD [–chown=<user>:<group>] <src1>... <dest>

与COPY使用格式一致，官方推荐使用COPY

VOLUME <path>

VOLUME ["<path1>", "<path2>"...]

定义匿名数据卷，在启动容器时如果忘记挂载数据卷，会自动挂载到匿名卷

WORKDIR <dir>

指定工作目录，会在构建镜像的每一层中都存在

ARG <name>[=<default value>]

构建参数，与ENV作用一至，不过作用域仅对Dockerfile内有效，构建好的镜像内不存在此环境变量

下面例举基于centos7构建sshd服务镜像的Dockerfile：

FROM centos

MAINTAINER This is my sshd.

RUN yum -y update \

&& yum -y install openssh\* net-tools lsof telnet passwd \

&& echo '123456' | passwd --stdin root \

&& sed -i 's/UsePAM yes/UsePAM no/g' /etc/ssh/sshd\_config \

&& ssh-keygen -t rsa -f /etc/ssh/ssh\_host\_rsa\_key \

&& sed -i '/^session\s\+required\s\+pam\_loginuid.so/s/^/#/' /etc/pam.d/sshd \

&& mkdir -p /root/.ssh && chown root.root /root && chmod 700 /root/.ssh

EXPOSE 22

CMD ["/usr/sbin/sshd", "-D"]

然后创建容器进行访问即可：

docker build -t captain\_rb/sshd:v1 .

docker run -d -p 2222:22 captain\_rb/sshd:v1

ssh -p 2222 localhost

附：

网址https://blog.csdn.net/lc15568886496/article/details/119607187

https://blog.csdn.net/Captain\_RB/article/details/115037179