链滴

# 我的 docker 探索之路

作者：xjtushilei

原文链接：https://ld246.com/article/1525961874479

来源网站：链滴

# 前言

docker处于了解并使用的情况，但使用也是在DevOps下使用的，只有一个宏观的了解，因此对docke的各种具体使用也不熟悉，所以决定探索一下，如果内存可以的话，将自己的云服务器全部docker化。

# 试验机介绍

本是在自己的虚拟机里，但是感觉不是很爽，虚拟机设置个16G内存，8C跑个小docker，对cpu和内的使用不能真的查看，想在生产环境里直接看一下效果。

服务器是阿里云的，今天刚买的一年的学生机，1C 2G 40G-SSD 1M，如下：


安装的操作系统是centos 7.3

# 安装 docker

```
yum update
curl -fsSL https://get.docker.com/ | sh
service docker start
```

设置开机启动：systemctl enable docker

在阿里云的“开发者平台”上进行换源，换成国内镜像，提高速度(脚本中的xxxxx是个人独有的，由里云生成的)。

```
sudo mkdir -p /etc/docker
sudo tee /etc/docker/daemon.json <<-'EOF'
{
  "registry-mirrors": ["https://xxxxxxxxxxxx.mirror.aliyuncs.com"]
}
EOF
sudo systemctl daemon-reload
sudo systemctl restart docker
```

测试helloworld：docker run hello-world

# 常见docker命令

列一下docker常用的操作，做个纪念吧。重点会用启动/停止/删除、看日志、进容器里就可以了。

- docker ps -a查看所有容器

- docker exec -it xxxxxx /bin/bash进入容器中并执行一个bash的shell（xxxxx是容器名字或id）
- docker stop $(docker ps -q)停用全部运行中的容器
- docker rm $(docker ps -aq)删除全部容器
- docker stop $(docker ps -q) & docker rm $(docker ps -aq)一条命令实现停用并删除容器：
- docker images | grep -E "(aaa|bbb)" | awk '{print $3}' | uniq | xargs -I {} docker rmi --force {}
除包含指定名称的Docker Image(含有aaa或者bbb的映像文件)
- docker logs -f xxxxxx查看容器的日志（xxxx：id或name，-f : 查看实时日志；-tail=10 : 查看最的10条日志。）
- docker runrun的参数有：

```
 -d, --detach=false         指定容器运行于前台还是后台，默认为false
  -i, --interactive=false   打开STDIN，用于控制台交互
  -t, --tty=false           分配tty设备，该可以支持终端登录，默认为false
  -u, --user=""             指定容器的用户
  -a, --attach=[]           登录容器（必须是以docker run -d启动的容器）
  -w, --workdir=""          指定容器的工作目录
  -c, --cpu-shares=0        设置容器CPU权重，在CPU共享场景使用
  -e, --env=[]              指定环境变量，容器中可以使用该环境变量
  -m, --memory=""           指定容器的内存上限
  -P, --publish-all=false   指定容器暴露的端口
  -p, --publish=[]          指定容器暴露的端口
  -h, --hostname=""         指定容器的主机名
  -v, --volume=[]           给容器挂载存储卷，挂载到容器的某个目录
  --cap-add=[]              添加权限，权限清单详见：http://linux.die.net/man/7/capabilities
  --cap-drop=[]             删除权限，权限清单详见：http://linux.die.net/man/7/capabilities
  --cidfile=""              运行容器后，在指定文件中写入容器PID值，一种典型的监控系统用法
  --cpuset=""               设置容器可以使用哪些CPU，此参数可以用来容器独占CPU
  --device=[]               添加主机设备给容器，相当于设备直通
  --dns=[]                  指定容器的dns服务器
  --dns-search=[]           指定容器的dns搜索域名，写入到容器的/etc/resolv.conf文件
  --entrypoint=""           覆盖image的入口点
  --env-file=[]             指定环境变量文件，文件格式为每行一个环境变量
  --expose=[]               指定容器暴露的端口，即修改镜像的暴露端口
  --link=[]                 指定容器间的关联，使用其他容器的IP、env等信息
  --lxc-conf=[]             指定容器的配置文件，只有在指定--exec-driver=lxc时使用
  --name=""                 指定容器名字，后续可以通过名字进行容器管理，links特性需要使用名字
  --net="bridge"            容器网络设置:
                        bridge 使用docker daemon指定的网桥
                        host    //容器使用主机的网络
                        container:NAME_or_ID  >//使用其他容器的网路，共享IP和PORT等网络资源
                        none 容器使用自己的网络（类似--net=bridge），但是不进行配置
  --privileged=false        指定容器是否为特权容器，特权容器拥有所有的capabilities
  --restart="no"            指定容器停止后的重启策略:
                        no：容器退出时不重启
                        on-failure：容器故障退出（返回值非零）时重启
                        always：容器退出时总是重启
                        unless-stopped: 除非人工停止，不然自动重启
  --rm=false                指定容器停止后自动删除容器(不支持以docker run -d启动的容器)
  --sig-proxy=true          设置由代理接受并处理信号，但是SIGCHLD、SIGSTOP和SIGKILL不能被理
```

- docker build -t yyyy xxxx构建docker镜像(yyyy:镜像的名字，xxxxx：dockerFile的所在目录)

# 安装mysql

安装最新版的mysql：同时设置容器名字，数据卷分离, 暴露端口，设置密码，后台启动，重启策略
远程镜像名字（不带版本号默认最新版）

docker run --name mysql -v /data/mysql:/var/lib/mysql -p 3306:3306 -e MYSQL_ROOT_PASSORD=你的默认密码 -d --restart=unless-stopped mysql:5.6.35

常用的几个就是我安装mysql时候用到的这些。

进入docker容器内部：docker exec -it mysql bash

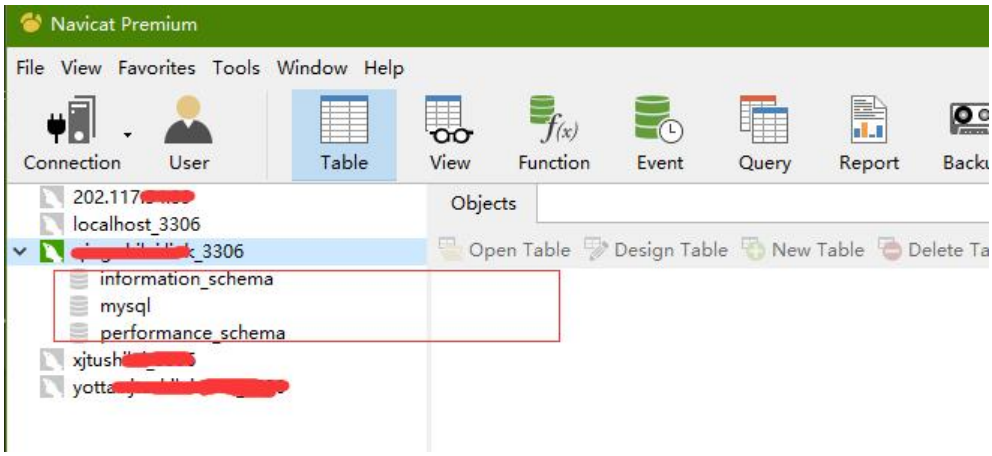
进入mysql的shell界面：mysql -uroot -p你的mysql用户密码

```
Warning: Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.6.35 MySQL Community Server (GPL)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> ^DBye
root@bf7a3fcfb42e:/# exit
```

想起来以前安装mysql好麻烦呀！docker使我感到开心。

在自己电脑上连接远程服务器，进行远程连接测试（首先你需要知道这是不安全的和生产环境绝对不许的，其次你需要打开防火墙）

如下，测试成功。




# SpringBoot 服务的部署

我写了个只有一个hi的web服务：https://github.com/xjtushilei/jenkins-test.git

并写了一个简单的 Dockerfile（下一节简单讲讲dockerFile的书写）

```
FROM openjdk:8-jdk-alpine
RUN mkdir -p /root/workspace/project
WORKDIR /root/workspace/project
```

```
COPY build/libs/*.jar app.jar
#RUN set -ex && ./gradlew build
#RUN cp build/libs/*.jar app.jar

ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/root/workspace/projec
/app.jar"]
```

其实三行就可以搞定，多写的主要是为了练习。

我分别尝试了在docker里进行构建和在docker外。（具体采取哪种方式要看DevOps了，自己开发的还是docker外吧，不然每次从maven中央仓库拉jar包很难受，即便换为国内的源；如果在docker内建的话，可以省略外部代码编译这一步，直接从代码到docker镜像，但是会延长发布时间）

我们先把代码拉下来，

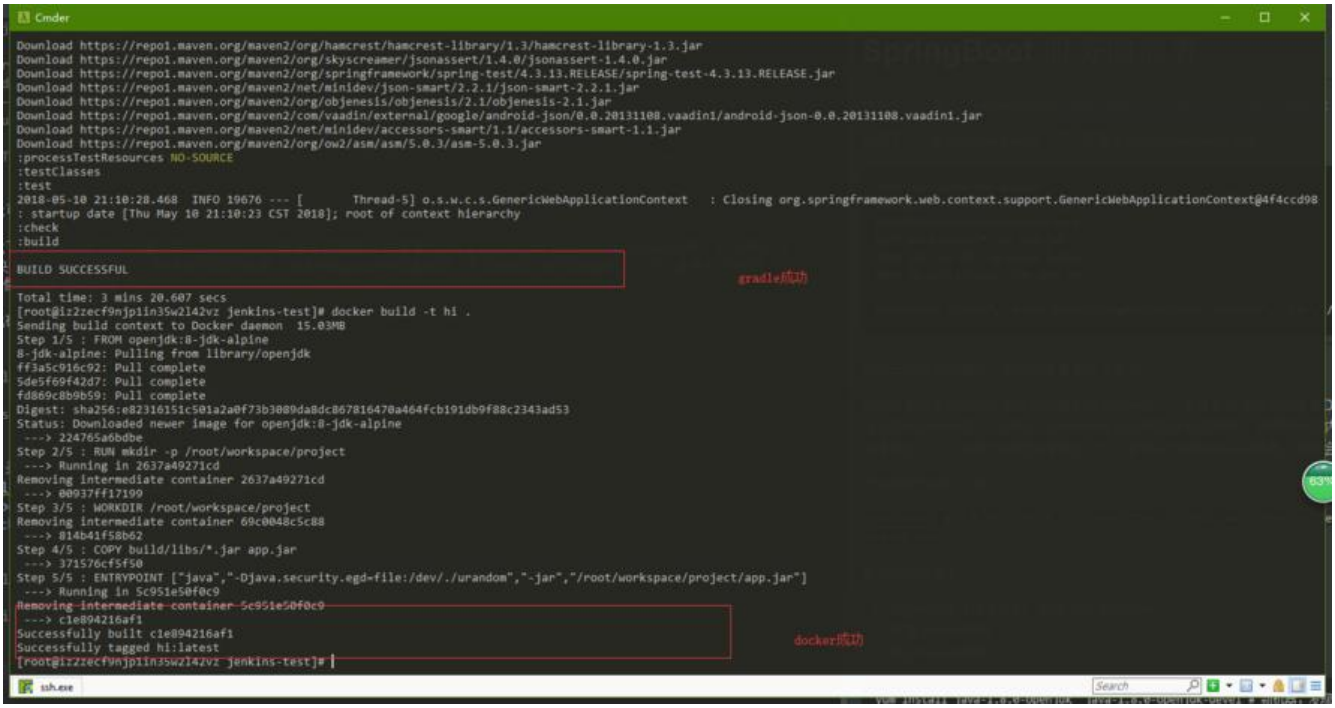
```
yum install git # 新机器，发现没装git，尴尬
git clone https://github.com/xjtushilei/jenkins-test.git
cd jenkins-test
```

在代码主目录下：

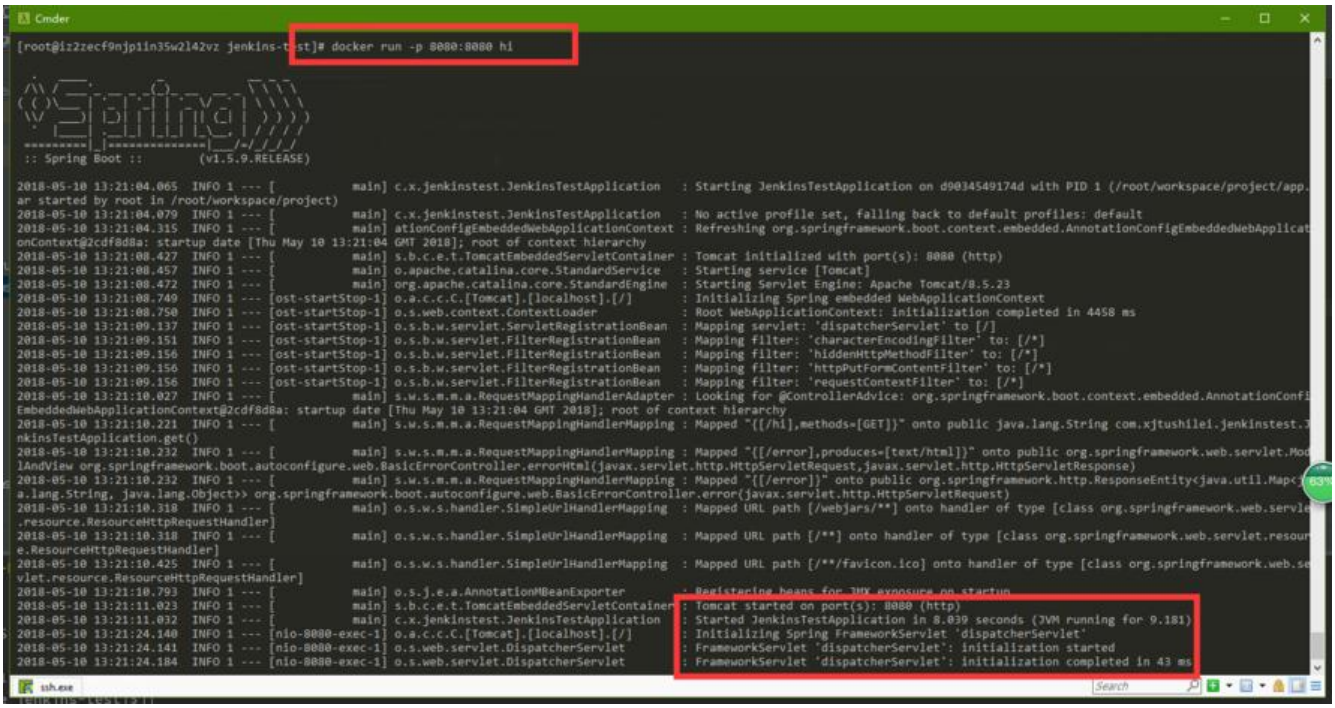
1. 用gradle编译打包代码，会生成可运行的jar
2. 构建 docker镜像
3. 运行docker镜像

```
yum install java-1.8.0-openjdk  java-1.8.0-openjdk-devel # 新机器，发现没装jdk，尴尬
./gradlew  build
docker build -t hi .
```

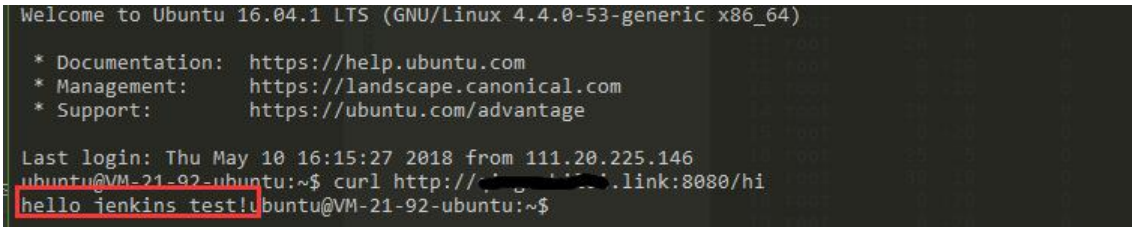
如下所示：构建我自己的docker成功了。




直接非后台运行：docker run -p 8080:8080 hi就看到了如下熟悉的界面：




远程测试成功：




到现在为止，我们也学会了如何发布自己的docker。我这里推荐使用阿里云的容器镜像服务，特别方！不用自己拉pull代码，直接用他们的服务检测git里代码的变化，动态更新镜像。

# Dockerfile 大致流程

- Dockerfile的第一条指令一般都是FROM,表示从一个基础镜像开始构建
- 执行一条命令对镜像做出修改
- 提交更新
- 基于本次更新，运行新的容器
- 继续执行下一条命令

如此反复执行……

在构建过程中每次生成一层新的镜像的时候这个镜像就会被缓存。即使是后面的某个步骤导致构建失，再次构建的时候就会从失败的那层镜像的前一条指令继续往下执行。

# DockerFile 的书写

DockerFile分为四部分组成：基础镜像信、维护者信息、镜像操作指令和容器启动时执行指令

```
#第一行必须指令基于的基础镜像
From ubutu
```

```
#维护者信息
MAINTAINER docker_user  docker_user@mail.com

#镜像的操作指令

RUN apt-get update && apt-get install -y ngnix
RUN echo "\ndaemon off;">>/etc/ngnix/nignix.conf

#容器启动时执行指令
CMD /usr/sbin/ngnix
```

下面讲一下DockerFile常见的指令

## FROM

格式为FROM或FROM:。

第一条指令必须为FROM指令。并且，如果在同一个Dockerfile中创建多个镜像时，可以使用多个FROM指令（高版本docker建议搭配as使用）。

## MAINTAINER

格式为MAINTAINER，指定维护者信息。

## RUN

格式为RUN 或RUN["executable", "param1", "param2"]。

前者将在shell终端中运行命令，即/bin/sh -c；后者则使用exec执行。指定使用其它终端可以通过第种方式实现，例如RUN ["/bin/bash", "-c", "echo hello"]。

每条RUN指令将在当前镜像基础上执行指定命令，并提交为新的镜像。当命令较长时可以使用\换行。

## CMD

支持三种格式

- CMD ["executable","param1","param2"]使用exec执行，推荐方式；
- CMD command param1 param2在/bin/sh中执行，提供给需要交互的应用；
- CMD ["param1","param2"]提供给ENTRYPOINT的默认参数；

指定启动容器时执行的命令，每个Dockerfile只能有一条CMD命令。如果指定了多条命令，只有最后条会被执行。

如果用户启动容器时候指定了运行的命令，则会覆盖掉CMD指定的命令。

## EXPOSE

格式为EXPOSE [...]。

告诉Docker服务端容器暴露的端口号，供互联系统使用。

这里仅仅是告诉，如果想给宿主机调用的话，需要run的时候-p 进行端口转发。

## ENV

格式为ENV 。 指定一个环境变量，会被后续RUN指令使用，并在容器运行时保持。

例如

```
ENV PG_MAJOR 9.3
ENV PG_VERSION 9.3.4
RUN curl -SL http://example.com/postgres-$PG_VERSION.tar.xz | tar -xJC /usr/src/postgress
& …
ENV PATH /usr/local/postgres-$PG_MAJOR/bin:$PATH
```

## ADD

格式为ADD 。

该命令将复制指定的到容器中的。 其中可以是Dockerfile所在目录的一个相对路径；也可以是一个UR
；还可以是一个tar文件（自动解压为目录）。

## COPY

格式为COPY。

复制本地主机的（为Dockerfile所在目录的相对路径）到容器中的。

当使用本地目录为源目录时，推荐使用COPY。这是和ADD的主要区别。

## ENTRYPOINT

两种格式：

- ENTRYPOINT ["executable", "param1", "param2"]
- ENTRYPOINT command param1 param2（shell中执行）

配置容器启动后执行的命令，并且不可被docker run提供的参数覆盖。

每个Dockerfile中只能有一个ENTRYPOINT，当指定多个时，只有最后一个起效。

## VOLUME

格式为VOLUME ["/data"]。

创建一个可以从本地主机或其他容器挂载的挂载点，一般用来存放数据库和需要保持的数据等。

## USER

格式为USER daemon。

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。

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，例：RUN groupadd -r postgres && useradd -r -g postgres postgres。要临时获取管理员权限可使用gosu，而不推荐sudo。

### WORKDIR

格式为WORKDIR /path/to/workdir。

为后续的RUN、CMD、ENTRYPOINT指令配置工作目录。

可以使用多个WORKDIR指令，后续命令如果参数是相对路径，则会基于之前命令指定的路径。例如

```
WORKDIR /a
WORKDIR b
WORKDIR c
RUN pwd
```

则最终路径为/a/b/c。

# docker的其他注意事项

- 和都必须是目录
- ``必须是容器中的绝对路径
- 路径如果不存在，执行完成之后，docker 会给宿主机创建该目录；``可以使用相对路径，但是相对并不是当前的工作目录，而是 /var/lib/docker/volumes/
- 如果只有一个路径，比如 docker run -it -v ，这种情况叫做匿名挂载，表示的是 container 中位置，宿主机会在`/var/lib/docker/volumes/`下随机创建一个目录与 container 中的对应
- 不管以何种方式 mount，容器销毁之后，由 -v 在宿主机上创建的目录不会销毁

- <host-path> 和 <container-path> 都必须是目录
- <container-path> 必须是容器中的绝对路径
- <host-path> 路径如果不存在，执行完成之后，docker 会给宿主机创建该目录；<host-path>可使用相对路径，但是相对的并不是当前的工作目录，而是 /var/lib/docker/volumes/
- 如果只有一个路径，比如 docker run -it -v <path> <image>，这种情况叫做匿名挂载，<image>表示的是 container 中的位置，宿主机会在 /var/lib/docker/volumes/ 下随机创建一个目录与 container 中的 <path> 对应
- 不管以何种方式 mount，容器销毁之后，由 -v 在宿主机上创建的目录不会销毁