# 编译环境Docker化

<mnstory.net>

编译代码，我喜欢操纵感强的，jenkins更适合构建，而不适合自己捣腾。

以前我们用虚拟机初始化一个环境，然后chroot，挺好，但是时常出现资源不够用的情况（很多人在一台HCI上创建虚拟机，不卡才怪），QEMU+KVM虚拟化的坏处是资源消耗太大，所以，用docker这种天生无太多牵挂的东西，性能会提高很多，而且更灵活（派生一个虚拟机和pull一个docker，感觉完全不同）。

## 修改问题

### 问题1 update grub失败

同事致力于构建自动化编译环境，他弄了一个编译镜像到200.200.1.230，但是有点问题，说是无法编译通过。

我下载下来仔细瞧瞧，错误来自于这个命令：

# /etc/kernel/postinst.d/zz-update-grub 3.10.0 /src/HCI5.5.2/build/pkg/root/boot/vmlinuz-3.10.0

grub-probe: error:

cannot find a device for /

直接exit 1了，跟踪了一下，应该是迭代/dev没有放通：

# ls /dev/mapper/

control

放通/dev后：

# ls /dev/mapper/ -l

crw-rw----. 1 root root 10, 58 Jun 7 01:11 control

lrwxrwxrwx. 1 root root 7 Jun 7 03:34 docker-253:0-2621888-22fa7426fc389e4b9ab7ed8ffaf54d495540f1511314f0c6c19c8e33e74667f3 -> ../dm-4

lrwxrwxrwx. 1 root root 7 Jun 7 01:47 docker-253:0-2621888-pool -> ../dm-3

lrwxrwxrwx. 1 root root 7 Jun 7 01:11 vg\_lcg-lv\_home -> ../dm-2

lrwxrwxrwx. 1 root root 7 Jun 7 01:11 vg\_lcg-lv\_root -> ../dm-0

lrwxrwxrwx. 1 root root 7 Jun 7 01:11 vg\_lcg-lv\_swap -> ../dm-1

但是还有错：

#/etc/kernel/postinst.d/zz-update-grub 3.10.0 /src/HCI5.5.2/build/pkg/root/boot/vmlinuz-3.10.0

Searching for GRUB installation directory ... found: /boot/grub

warning: grub-probe can't find drive for /dev/mapper/docker-253:0-2621888-22fa7426fc389e4b9ab7ed8ffaf54d495540f1511314f0c6c19c8e33e74667f3.

grub-probe: error: cannot find a GRUB drive for /dev/mapper/docker-253:0-2621888-22fa7426fc389e4b9ab7ed8ffaf54d495540f1511314f0c6c19c8e33e74667f3. Check your device.map.

网上的资料，比较少，特别是在不能上GOOGLE的年代，某度基本上找不到有价值的信息，strace能看出一点问题，但最重要的得靠脑补，这么想，错误是由于grub安装导致，grub安装的时候引用的是原/boot分区，看来docker版本里面的/boot分区有点问题，那就用外面的吧，放通，另外还要加上权限--privileged，可以。

另外，上诉放通办法，只在device mapper情况下有效。

### 问题2 svn co的时候遇到编码问题

先修改字符编码，export LC\_CTYPE="zh\_CN.UTF-8"

然后locale-gen 生成/etc/locale.gen 文件

然后vi /etc/locale.gen，移除zh\_CN.UTF-8前面的注释

再次locale-gen

## 制作镜像

VT编译环境最开始是我和老杨设计的，自认为优雅大方美丽。

最开始的原则是：每次打包重新check out，一次编译通过。

但是原则就是用来打破的，自从第一次有人check in代码，让编译环境不能一次编译通过后，几年来这原则就陆陆续续被破了几次，新人一边在心里暗骂这是什么鬼东西一边一筹莫展询问群上的老鸟为啥编译不过？老鸟很多是知道原因的但是不会去修改，毕竟人性是懒惰的，另外自己的任务还要靠加班呢，再其次就是，凑合用也没啥问题，至于新人，也可以这么理解，等你多修炼几年，自然能编译通过，目前还是实力不够。

对应打包环境，我后来调整了几次，特别是针对公共库的使用上，很多原生的库，不需要每次都编译消耗CPU消耗时间，整理到3party下二进制发布，针对某些需要多次编译通过的，整理为一次编译通过，不过有些模块还是第一次创伤后的样子，比如usbredir库依赖问题，本次就遇到了，额，怎么解决的？不细说了，反正，编译都搞不定，也不指望你还能搞定其他东西。

修改好后提交到公司服务器上：

do ~ # docker commit compilevt compilevt\_img

81e9c9bc43d41d7fc40a68c4c00d00eac0f6c66c6dc6427a46e901dced1d76d2

do ~ # docker tag compilevt\_img 200.200.1.230/compiler/hci5.5

do ~ # docker push 200.200.1.230/compiler/hci5.5

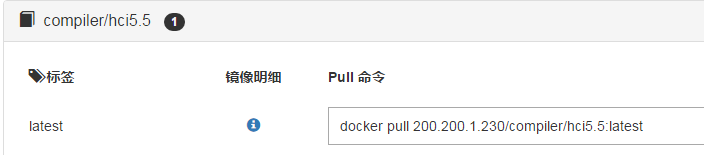
The push refers to a repository [200.200.1.230/compiler/hci5.5] (len: 1)

81e9c9bc43d4: Image already exists

bc633218bc76: Image successfully pushed

cac7d4b0ce11: Image successfully pushed

此处略去一千字……



为什么略去一千字，因为我是在前人基础上做的镜像， layer太多，作为非处女的某星座，我也对细节紧扣得要命，每每想到这么多layer要一遍一遍解析，就替CPU感到累。

这是原因一，第二个非常不好的是，layer太多，我都不知道里面的某一层是否放了源码，必须把私密的东西清除干净，毕竟，你怎么知道镜像不会外泄？放到公司服务器上的，还得考虑公司大了什么品种的鸟都有。

于是我准备重新制作镜像，把之前的export一下，顺便加点工具，比如cmake等编译工具，再import，减少layer，删除敏感信息，特别是，镜像里面不能有源码和密码。

docker export --output="/home/compiler.tar" 6b3222038e7f

cat /home/compiler.tar | docker import - compiler\_img

docker tag compiler\_img 200.200.1.230/compiler/hci5.5

docker push 200.200.1.230/compiler/hci5.5

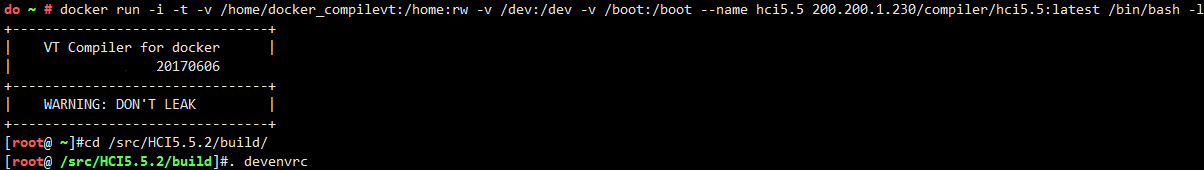
这次镜像下载大小为416MB，解压后占用1.19G，比export之前的9.813G小很多。

## 使用

本着无状态原则，编译环境需要运行的软件和源码之间应做分离，做到可用同一个容器编译不同的代码，容器删除了，源码可落地，所以，建议对源码路径做单独volume，例如我这里映射的/home分区，运行为：

mkdir -p /home/docker\_compilevt 2>/dev/null

docker run -i -t --privileged -v /home/docker\_compilevt:/home:rw -v /dev:/dev -v /boot:/boot --name hci5.5 200.200.1.230/compiler/hci5.5:latest /bin/bash -l



实测，打包出来的镜像可行，而且性能比在虚拟机上跑要高很多，不信你可以试试。

2017/6/7