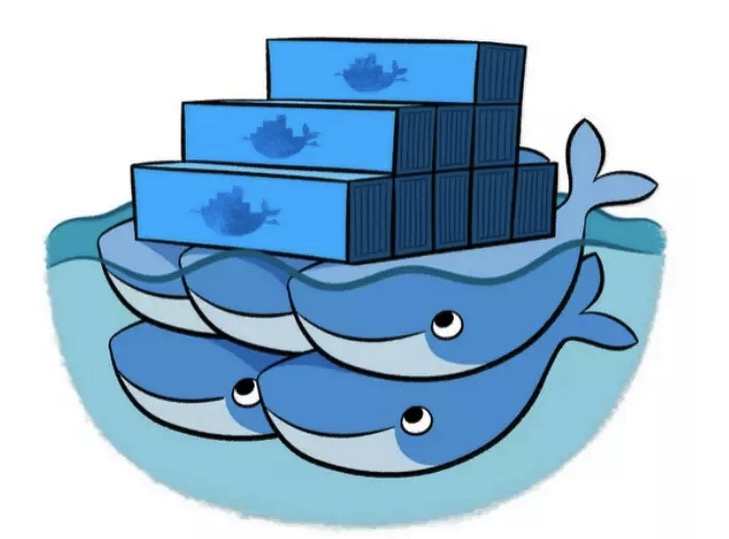
**七牛技术实践：如何使用DOCKER构建项目**

[牛小七](http://blog.qiniu.com/archives/author/mktqiniu)[2015年4月1日](http://blog.qiniu.com/archives/2068)发布在 [技术分享](http://blog.qiniu.com/archives/category/share)

[](http://blog.qiniu.com/wp-content/uploads/2015/04/11.jpg)

从2013年3月发布第一个版本，仅仅两年时间，Docker已经成为最炙手可热的开源项目之一，它也受到越来越多的开发与运维人员的亲睐。

在由InfoQ主办的QClub之Docker专场上，七牛全栈架构师李兆海分享了七牛如何使用Docker来构建项目。项目的构建经常会困扰开发者，因为构建流程可能会依赖编译环境，并且项目还可能会有外部依赖。李兆海分别介绍了七牛在使用Docker构建项目的几种方案，并分析了各个方案的优缺点。

**为什么构建项目是个问题？**

大家在工作中会发现项目构成是一个比较复杂的过程，它会依赖很多东西，它需要很多编译语言，需要依赖到包的环境和其他外部的相关工作，像测试集成等等，这些都是相对比较复杂的。比如要构建一个C的库的话会有很多依赖，有很多东西需要配置。管理这些依赖的情况下有很多工具，工具会有自己不同的特点需要去考虑。

**一、编译环境**

* 作为编译环境，首先会涉及到操作系统的问题，在不同的操作系统下编译会有微妙的差别，社区本身会有很多工具屏蔽这个差距，即便屏蔽的再好也会有很微妙的差距。
* 项目管理也是需要考虑的，比如低版本的问题，有些功能不一样，还有团队比较大的话，大家用的版本管理工具可能不太一样，这个现在不是太突出了，但是以前项目管理的问题还是比较突出的。
* 编译器也属于编译环境里的问题，一般公司内部会统一一个官方的编译器，但是开发者在自己使用的时候有可能偏离公司内部编译器的版本。
* 目标环境也是大家经常需要考虑的问题，一般来说的话有可能你的开发环境和实际的生产环境不一致，这个时候也会造成很微妙的差异问题。

**二、外部依赖**

* 构建项目需要确认依赖包的版本，这方面有些语言有包管理工具可以使用。不过包管理工具并不能做所有事精。比如七牛内部经常使用的Go语言，这个会自带一个包管理，但是不能自定义，有一些需要靠自己编译环境解决的问题。
* 如果公司大的话还会有项目之间的依赖，这个也是一些需要去考虑的，如何从别的项目组来拿取相关的版本依赖的问题。

**三、持续集成**

* 构建自动化：构建的过程要想自动化，需要保证编译行为和本地开发一致（环境一致，结果一致）。自动化构建失败时的清理工作。
* 部署自动化：部署自动化要保证依赖项目也要部署一致。部署失败时的回滚。

**Docker是如何解决这些问题的？**

“Build,Ship and Run Any App, Anywhere.”

大家可能更多讲的是run和ship，我更多的是讲build，而且这个目的是在任何地方都可以对它做build。

**一、编译环境**

编译环境也是一种环境，Dockerfile作为一个Docker的封装语言，它是在描述这个环境本身，而且非常幸运的是Dockerfile是非常好的纯文本文件，所以它很容易和平时使用的版本控制工具集成在一起，然后对整个环境做一个版本控制。而且由于Docker本身是一个自封装式的东西可以做编译环境和运行时的环境完全一致。这对屏蔽差异是一件非常好的事情。

**二、外部依赖**

包依赖属于环境的一部分，其实如何做包的管理，就相当于说你在管理相关的具体的环境。项目间的依赖会有另外的方法，一种就是把项目之间做成一个单独的包的依赖，这种方法等同于之前处理环境下的环境内部包的依赖。另外一种方法就是把所有的项目合并在一个大的项目，这样的话就不会出现项目间的依赖，其实都是在一个项目内部。

**三、持续集成**

Docker本身是一个统一环境的能力，所以从持续集成的工具来看的话，它不需要涉及到具体的构建的内部和部署的内部，只需要能够构建Docker的一些功能就可以。所以它其实是在这个层面上是可以简化编译脚本和简化部署脚本编写的。

**使用Docker构建的尝试**

[](http://blog.qiniu.com/wp-content/uploads/2015/04/21.jpg)

这是很简单的Go的程序。它首先会引入两个包，一个是官方包，一个是外部包，它需要包管理依赖工具来引用到。这个数主函数，它做的就是先建一个http，然后通过官方里的App功能把这个启动起来，然后在1234端口打开。可以看到这个程序没做太多事情，主要演示的就是包工具管理依赖的事情。

[](http://blog.qiniu.com/wp-content/uploads/2015/04/31.jpg)

这是做的第一次尝试，可以看到这个目录结构是这样的，这个是按照Go的官方习惯的编写方式，会把文件放在叫做src下面，hello可以看作一个是项目表，main就是文件。

[](http://blog.qiniu.com/wp-content/uploads/2015/04/42.jpg)

这个是刚才目录里的Dockerfile。它首先引用了一个Golang，这个是Docker官方提供的，它除了提供这个还提供其他语言的，Java都有。它做的事情很简单，首先把src目录加到main里，然后拉去hello模块里的依赖。从Golang，当把src目录加进去之后就会把它作为Golang执行时的目录。这里不用把前面的前缀都写，只需要写在Src目录下面的名就可以了，就是hello。

**优缺点**

* 优点：同步更新项目依赖；结构简单，容易维护。
* 缺点：无法对依赖做cache，编译耗时；不同项目之间重新编译依赖。

[](http://blog.qiniu.com/wp-content/uploads/2015/04/51.jpg)

第二次尝试，这个结构跟刚才一样，这里和刚才有一个最大的区别，它就是说把刚才的依赖包直接手写到了Docker文件里，而且这个依赖包会写在，把src加入到整个Docker的地址之前，就是每次先执行这句话，然后再把所有的原密码上传设置。这个好处就是在于这句话执行与否与互相间的修改不相关的。如果你重复执行Dockerfile，可以在这里面直接cache，这句话只要在第一次运行一遍直接走到下面，这就会增加编译的速度。如果想更新这个包的话，可以减cache命令，它就会重新拉取依赖，这个依赖就是一个可以被cache目标。

**优缺点**

* 优点：依赖会被Cache，提高速度；结构简单，容易维护。
* 缺点：手写依赖，不能同步更新项目的变化；不同项目之间重新编译依赖。

[](http://blog.qiniu.com/wp-content/uploads/2015/04/62.jpg)

第三次尝试，这个可以看到整个目录结构开始变得比较复杂了，首先最大不同就是编译变成了两个脚本。Dockerfile没有变，在src会出现一个github.com。Dockerfile只是直接把一个hello拷到了Docker里，而没有做其他的事情。

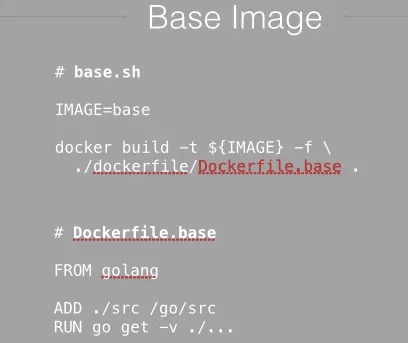
**优缺点**

* 优点：依赖会被保存在本地，提高速度；依赖同步更新。
* 缺点：不同编译环境保存的依赖可能不一致；依赖侵入项目，造成干扰；gitignore，但是很麻烦。

**使用Docker进行构建**

[](http://blog.qiniu.com/wp-content/uploads/2015/04/71.jpg)

这是我们现在在使用的一些方法，是我们最终在使用的。首先可以从目录结构来看，它会比刚才第三种方法变得更加复杂，首先它会有两个编译的脚本，这个应该是build sh.sh，hello.sh，我们写程序的时候知道，如果有问题的话可以单独抽象出一层来解决问题，这个就是把一个基础的依赖和真正构建程序的Docker分成两个Docker，base这层是在做依赖，而hello会基于base做构建。

[](http://blog.qiniu.com/wp-content/uploads/2015/04/81.jpg)

这个是来介绍base Image，它所做的事情首先是起名字，同时会指定。下面这个Dockerfile就是base，它做的就是从Golang程序里作为一个基础隐秘，把src加入进去，这是Go的一个配置方法，相当于它会拉取所有项目依赖。这样做完以后归做base Image有项目所有依赖的包。

[](http://blog.qiniu.com/wp-content/uploads/2015/04/91.jpg)

这是整个项目的所有依赖，这里hello.sh是构建项目的脚本，它跟刚才的最大区别是它构建的配置文件发生了变化，这是指定了一个hello的名字。而下面是hello的cache文件。这个地方会发现，把src重新加入到image，这样做就是当helloimage有变化的时候，它还是需要重新覆盖到image。

**优缺点**

* 优点：依赖会被cache在本地，提高速度；依赖同步更新；依赖可以被多个项目共享。
* 缺点：多维护一个image。

以Docker为代表的容器技术日趋火热，乘着云计算之风，这个技术新星将引领新一轮的技术热潮。