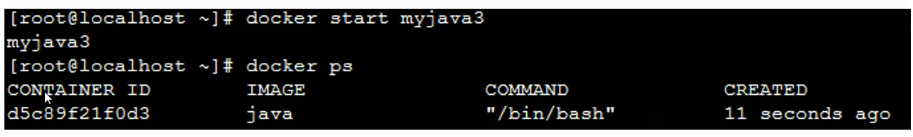


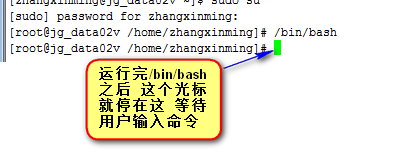
这个是使用docker创建了一个容器

现在启动这个Container

要执行的命令 就是 /bin/bash



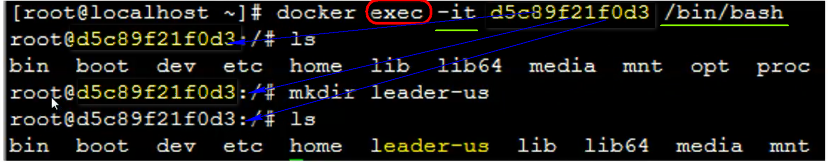
使用docker ps查看 发现这个镜像是存在的 为什么呢？



这样 这个一直是前台执行的没有完成的命令 所以 docker ps可以查询到这个容器的存在

启动之后在前台执行

然后使用exec链接到这个Container上面 执行一个/bin/bash

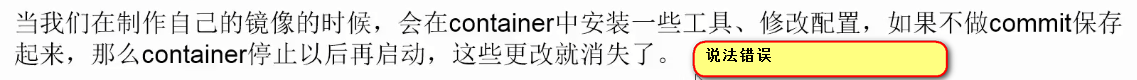


【执行之后 可以看到 从[root@localhost ~]变成了root@d5c89f0d3:/ 说明进入了docker的内部容器】

进来之后 执行ls 有下面的目录

执行mkdir leader-us

然后 我们使用stop或者kill干掉这个容器 那么 生成的这个leader-us的目录是依然存在的 这就证明了上面的说法是错误的



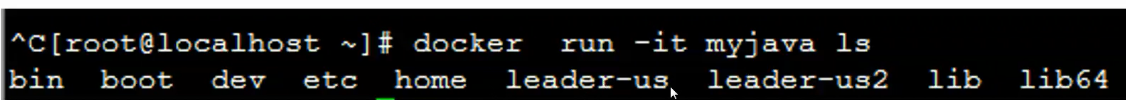
使用commit的确可以创建一个新的image



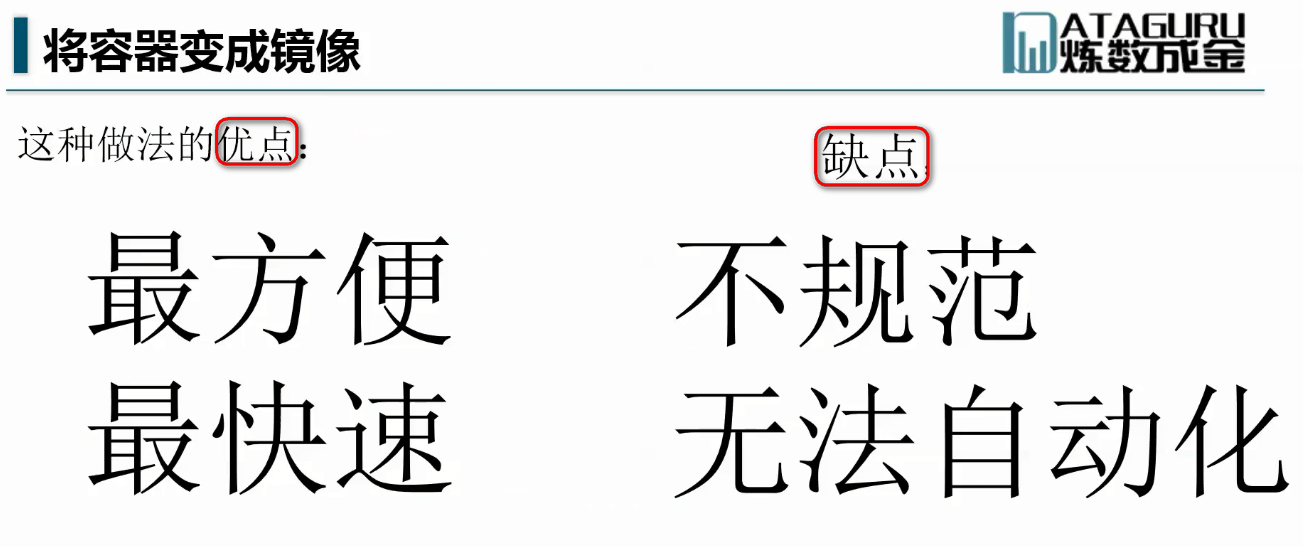
使用Commit创建一个镜像 镜像名就是myjava

创建完之后 因为在创建镜像的时候 没有指定版本号 所以 使用docker images就可以看到TAG这一列就是latest

然后使用docker run这个命令创建并启动这个容器 就会看到



优势 和 劣势



完全手工的方式 依赖于某一个镜像 所以无法自动化 ---- 这个和Docker生命周期矛盾的

适合本地开发测试验证

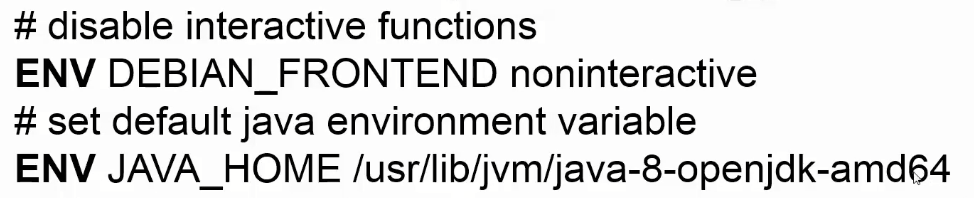
============

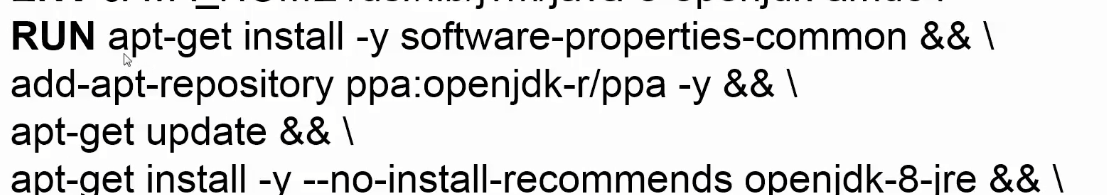
Buildfile --- 创建docker镜像最标准的做法

维护者 ---- 指的是这个image的制作人



增加一些环境变量

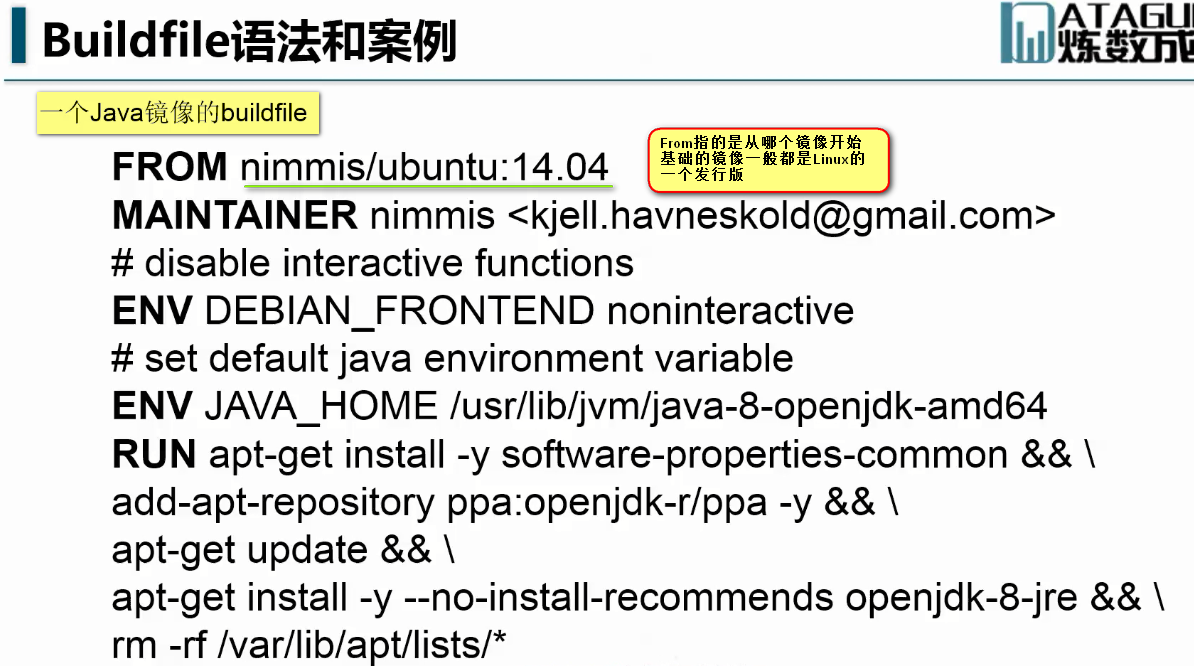
 环境变量是传递参数的一个重要的办法

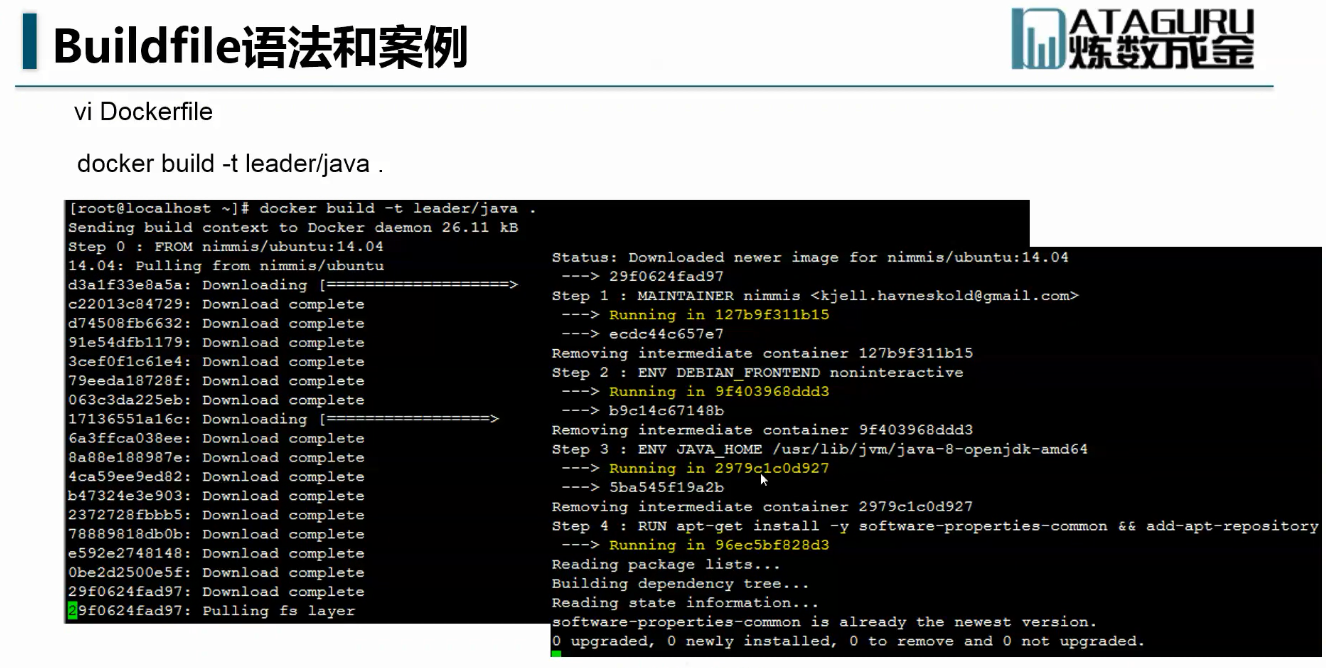


RUN就是执行linux中的一些常用的命令 install就安装一些常用的包

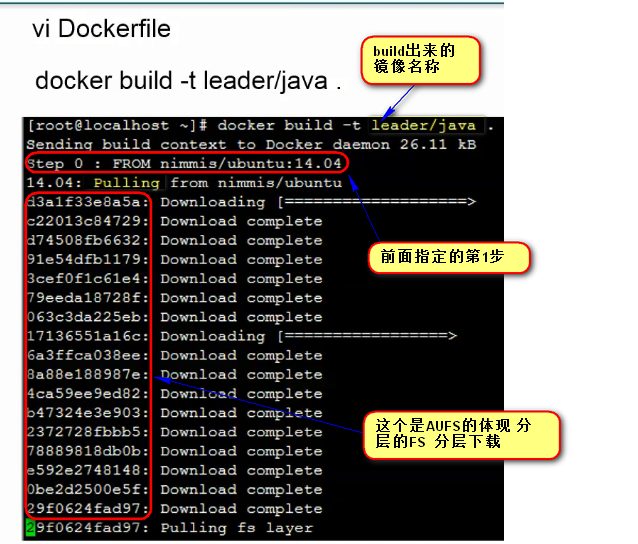
 删除安装包

【\就是多行指令 一次执行完成】

 这个是写的一个build文件的示例



 leader/java 表示build出来的镜像的名称



**.表示在当前目录找对应的buildfile**



**注意到 每一行的指令都在一个容器中执行 这个非常重要**

**如果在CentOS上执行 没有apt-get 这个指令**

**但是 这个是可以执行成功的 ---- 因为刚才说了 每一步都是在一个中间的容器中执行的 ----- 这样 因为是From**  **最最基础的镜像就是Ubuntu 那么这个镜像就是Ubuntu ---- 而不是在宿主机centos上执行**

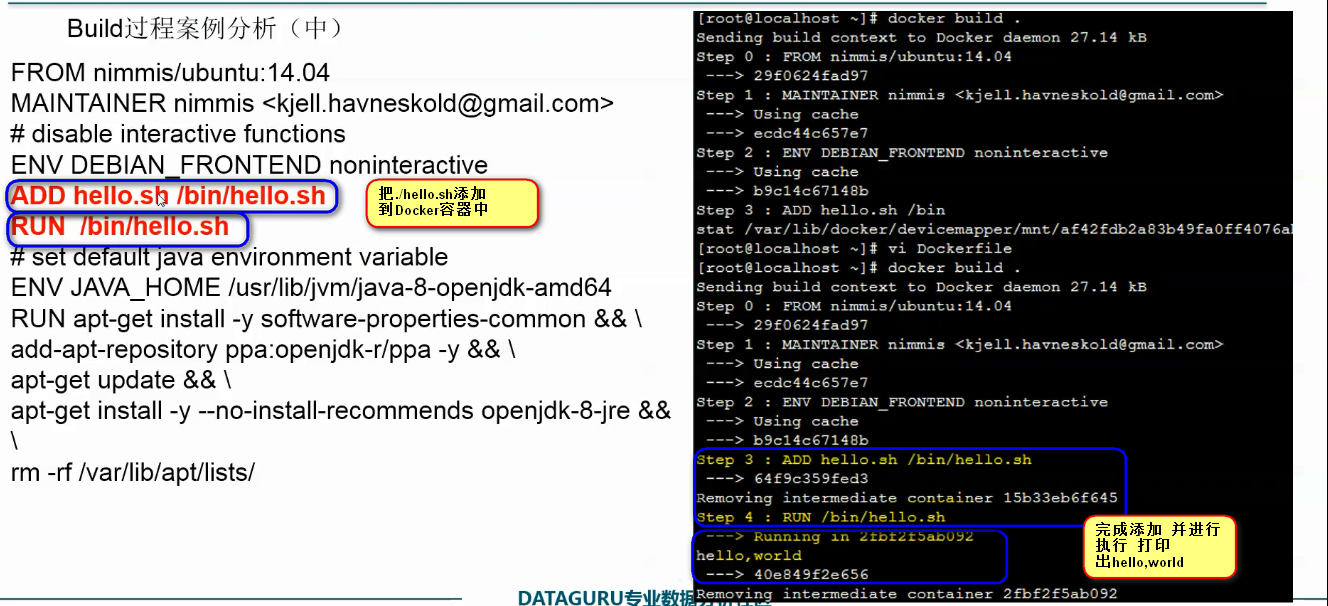
**所以 这个buildfile中的run可以执行成功**

**每一步都是在中间Docker中执行 这个是理解build的核心** ----------------

现在 我希望执行一个hello.sh这个指令 向屏幕输出hello world



因为执行不是在宿主机 而是在容器内 ----- 所以 把这个文件加进去

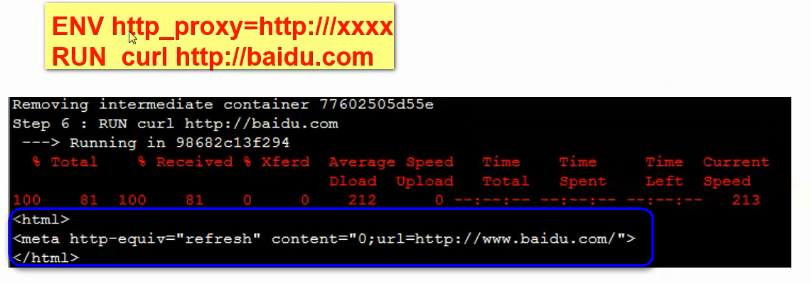


编译过程所有的指令 都是在容器内执行的 和你的宿主机无关

比如我的机器需要代理才能上网 然后 需要RUN curl <http://baidu.com>



就是无法连接 虽然安装docker的宿主机配置了代理 但是 我的Docker容器内部是没有设置代理的 所以 我要在运行这个Curl命令之前添加一个环境变量



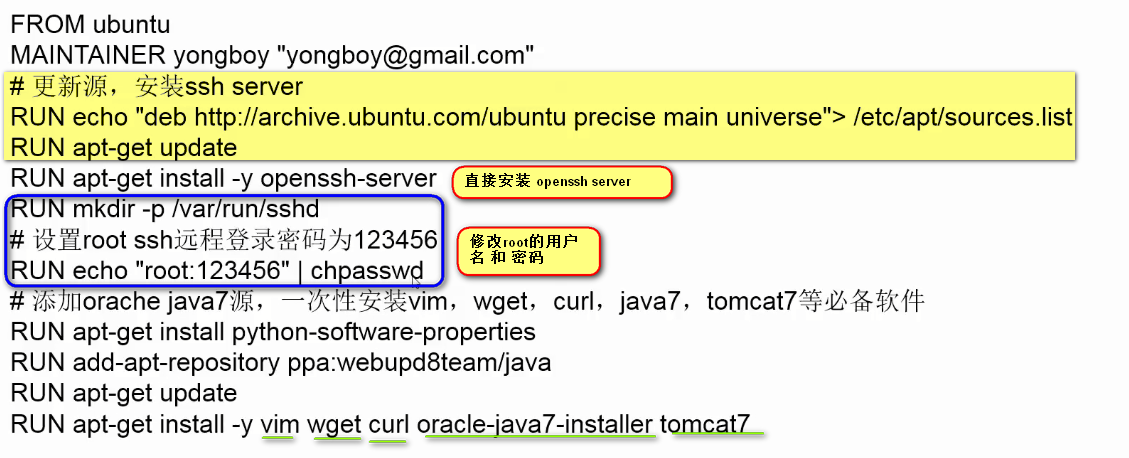
--------------------- 复杂的案例实战

以Ubuntu为基础的镜像

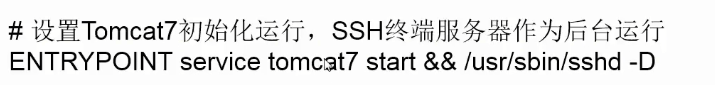




 -y表示自动安装 不用回车







-D 会作为Daemon进程启动 输出一段话就结束了 这样就不正常了 必须前台运行 ---- Docker中的程序必须在前台执行

所以 可以在后面添加一个tail –f 输出日志 这样让一个命令前台执行 就没问题了

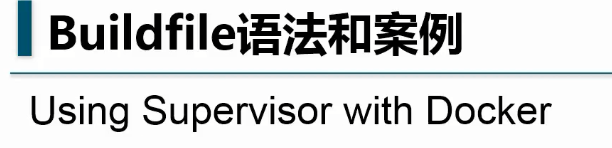
=========

一般情况下 一个docker容器里面允许执行一个命令 有时候 需要启动多个进程

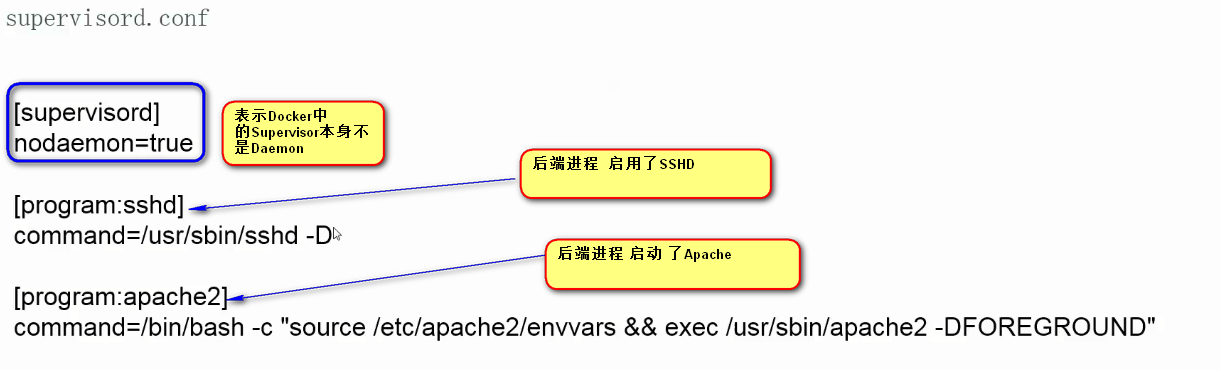
**多进程的做法：某一个进程在前台执行 剩余的多个进程后台执行**

这种做法比较麻烦

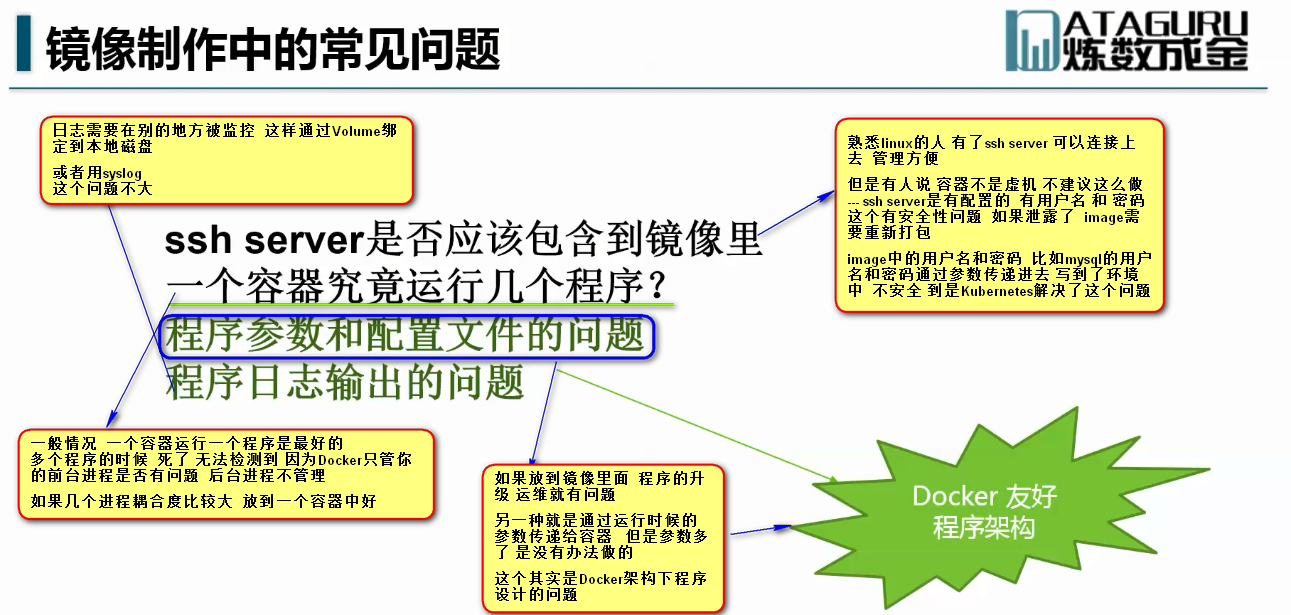
使用Supervisor



这个就是 把Supervisor作为Docker容器的前端进程 执行 然后 你真正要执行的命令 放到[program:xxxx] 作为Daemon进程执行



Supervisor的好处就是 某一个进程死掉了 supervisor会把这个进程重启 ----- -所以 多个进程 Supervisor是一个最佳实践

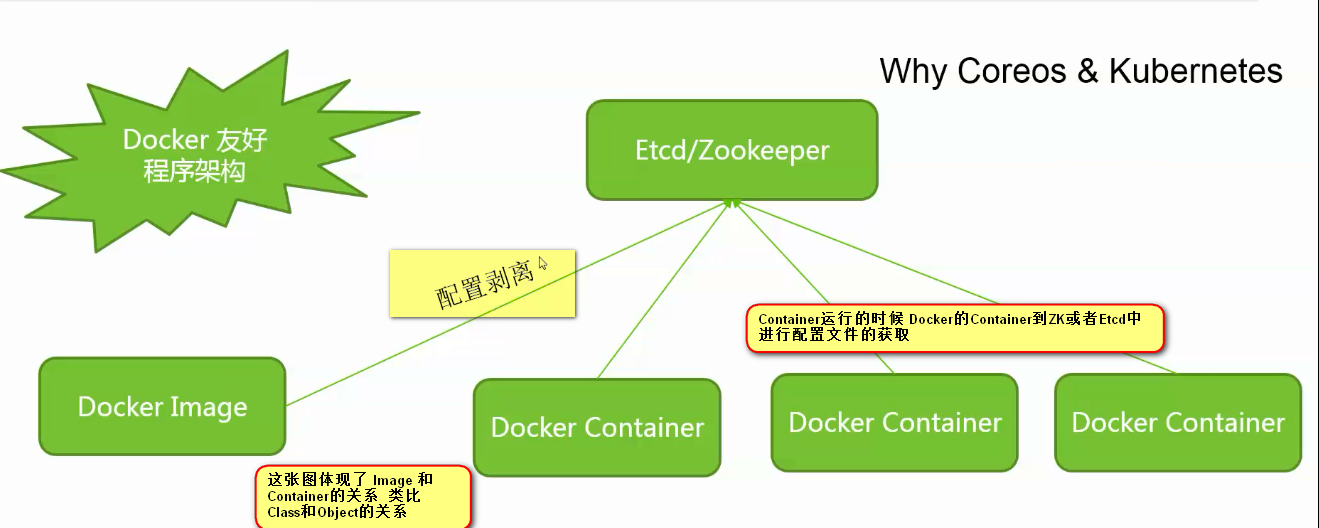


现在看看Docker友好的程序架构

把程序参数 或者配置文件放到Zookeeper/Etcd中

ETCD是Docker生态圈的新技术 是Zookeeper的升级 是Google的

是KV的存储信息



这个是Docker友好的程序架构 ---- 也是Coreos和Kuberneters推荐的架构

Kubernetes把Etcd作为资源存储的核心 最终调度各种服务