

Docker持续部署图文详解（附实战视频） | 高效运维最佳实践06

2015-07-21 萧田国、张春源 高效运维

前言

关于Docker的文章铺天盖地，但精品文章往往翻译居多。都说Docker天生适合持续集成/持续部署，但同样，可落地、实际可操作性的文章也很少见。

基于这些情况，虽然我们专栏定位为运维管理性文字，但本篇是个特例，实操性的案例讲解——JAVA项目如何通过Docker实现持续部署（只需简单四步），即：

- 开发同学通过git push上传代码，经Git和Jenkins配合，自动完成程序部署、发布，全程无需运维人员参与。

这是一种真正的容器级的实现，这个带来的好处，不仅仅是效率的提升，更是一种变革：

- 开发人员第一次真正为自己的代码负责——终于可以跳过运维和测试部门，自主维护运行环境（首先是测试/开发环境）。

本文是cSphere Docker实战视频第二讲的文字版，本文联合作者@张春源同学（任职cSphere）即为视频主讲人，关于更多系列视频，详见<https://csphere.cn/training>。

福利：点击文末的“[阅读原文](#)”即可手机欣赏本文对应的实战视频哦。

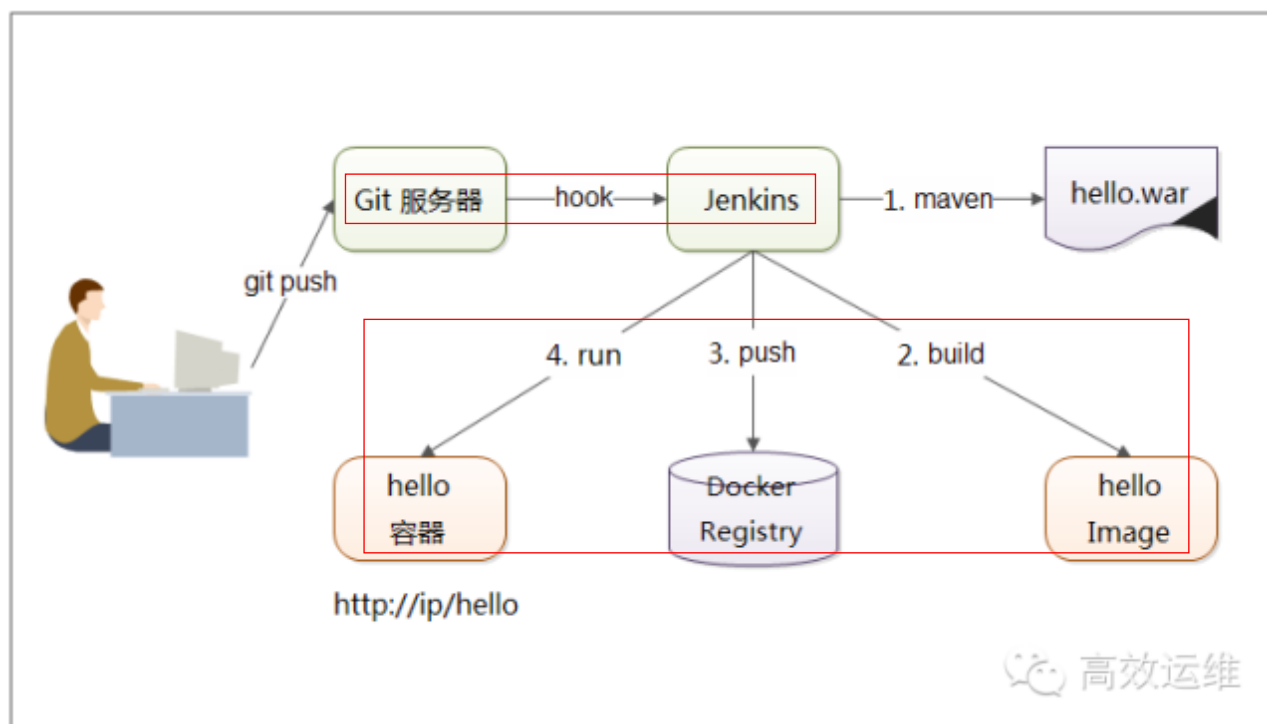
难者不会，会者不难。通过简单的4个配置，即可优雅地实现持续部署。本文依惯例放上目录，请享用：

1. 持续部署的技术思路
2. 效果展示
3. 配置Git和Jenkins联动
4. 配置Jenkins自动更新代码
5. 效果图文详解
6. FAQ

好吧，我们正式开始。

1. 持续部署的技术思路

在本例中，假设我们JAVA项目的名称为hello。简要的技术思路如下。



本案例中假设代码托管在git.oschina.com上，Jenkins和Docker Registry（类似于yum源）各运行在一个Docker容器中。JAVA项目自己也单独运行在一个叫hello的容器中。

本文采取的持续部署方案，是从私有的Docker Registry拉取代码，然后通过重建image来实现。这里Jenkins处于中心位置。就像长臂猿，在接收到Git的请求后，通过远程调用服务器Shell脚本，完成几乎所有功能。

另外，有些变通的方案，把代码放在宿主机上，让容器通过卷组映射来读取。这种方法不建议的原因是，将代码拆分出容器，这违背了Docker的集装箱原则：

这也导致装卸复杂度增加。从货运工人角度考虑，整体才是最经济的。这样，也才能实现真正意义的容器级迁移。

或者说，容器时代，抛弃过去文件分发的思想，才是正途。本文最后的问答环节对此有更多阐述。

容器即进程。我们采用上述方案做Docker持续部署的原因和意义，也在于此。容器的生命周期，应该远远短于虚拟机，容器出现问题，应该是立即杀掉，而不是试图恢复。

2. 效果展示

本文最后实现的效果，究竟有多惊艳呢？且看如下的演示。

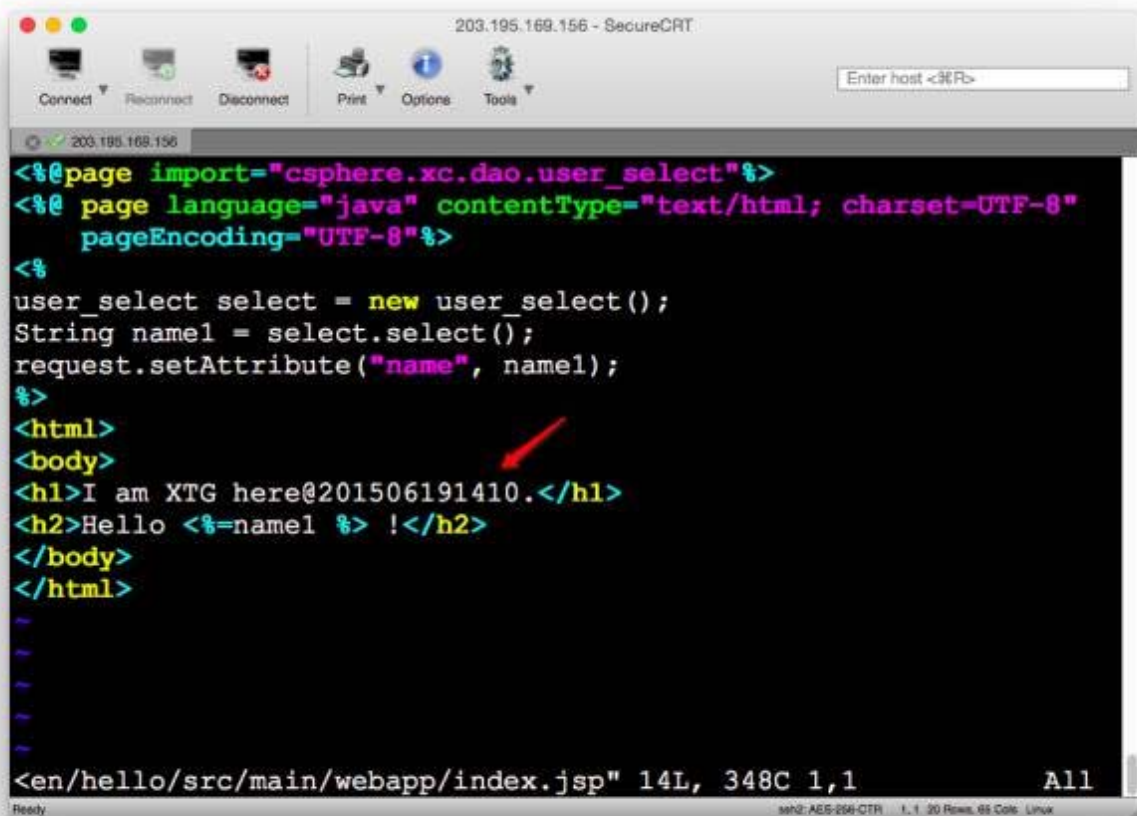
2.1 程序代码更新前的效果

我们以时间戳来简洁、显式的表述程序更新情况。



2.2 提交程序代码更新

本例中，我们把首页的时间戳从201506181750，修改为201506191410（见如下）。



```
203.195.169.156 - SecureCRT
Connect Reconnect Disconnect Print Options Tools
203.195.169.156
<%@page import="csphere.xc.dao.user_select"%>
<%@page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%
user_select select = new user_select();
String name1 = select.select();
request.setAttribute("name", name1);
%>
<html>
<body>
<h1>I am XTG here@201506191410.</h1>
<h2>Hello <%=name1 %> !</h2>
</body>
</html>
<en/hello/src/main/webapp/index.jsp" 14L, 348C 1,1 All
Ready ssh2: AES-256-CTR 1, 1 20 Rows, 95 Cols Linux
```

高效运维

2.3 上传新代码到Git

顺序执行如下操作，输入正确的git账号密码。

```
git add *
git commit -m "Modified the front-end display content"
git config --global user.email "xiaotianguo@foxmail.com"

git push
```

然后，输入git账号密码，然后等待神奇的一幕... 高效运维

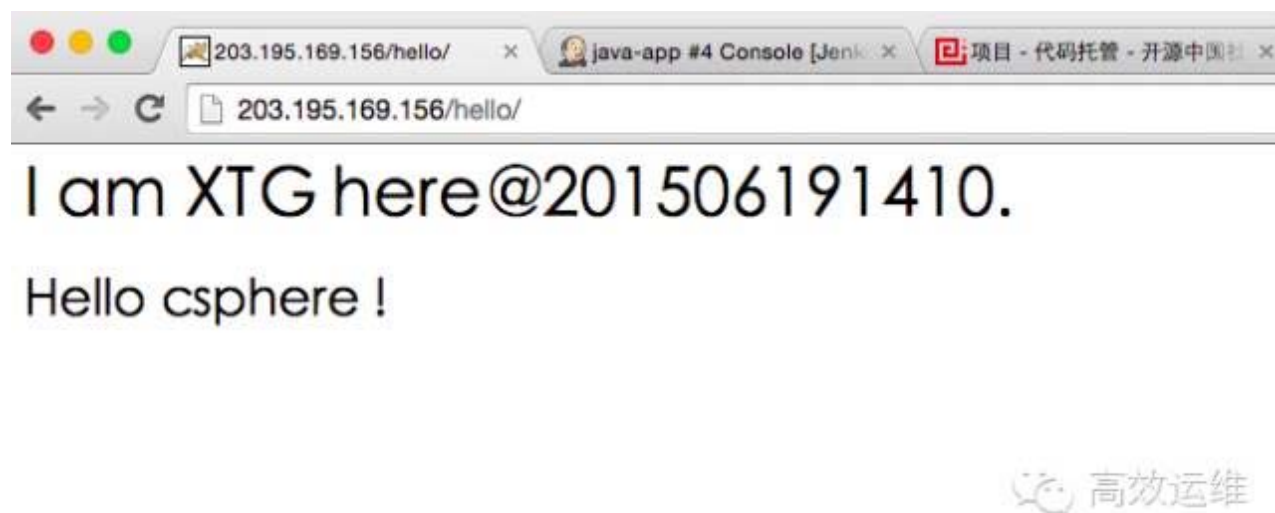
然后呢？

然后什么都不用做了。端杯茶（如果不喜欢咖啡的话），静静地等待自动部署的发生，旁观一系列被自动触发的过程，机器人似的运转起来（请容稍候再加以描述）。

为什么需要3~5分钟？只是因为本案例中的JAVA项目，需要从国外download Maven程序包，以供Jenkins调用和编译JAVA。正式应用环境中，可以把Maven源放在国内或机房。如果仅仅需要对PHP项目做持续部署，那就更快捷了。

2.4 查看代码更新后的效果

在静静地等待几分钟后，新的代码确实已经自动部署完毕。



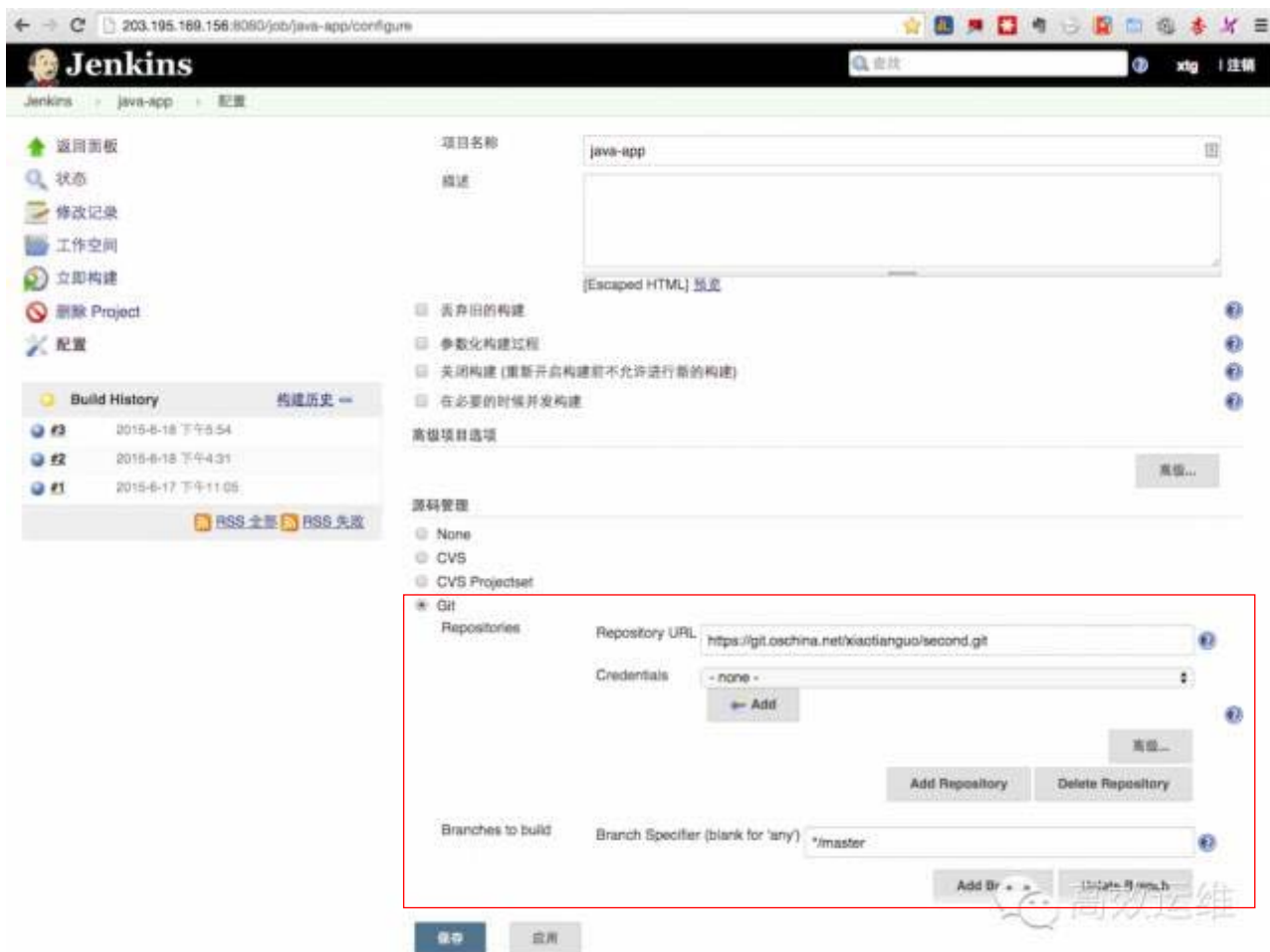
那么，这一切怎么实现的呢？很复杂么？不然。只要按照如下几步，便可快速实现哦。

3. 配置Git和Jenkins联动

这个过程主要分为如下三步。

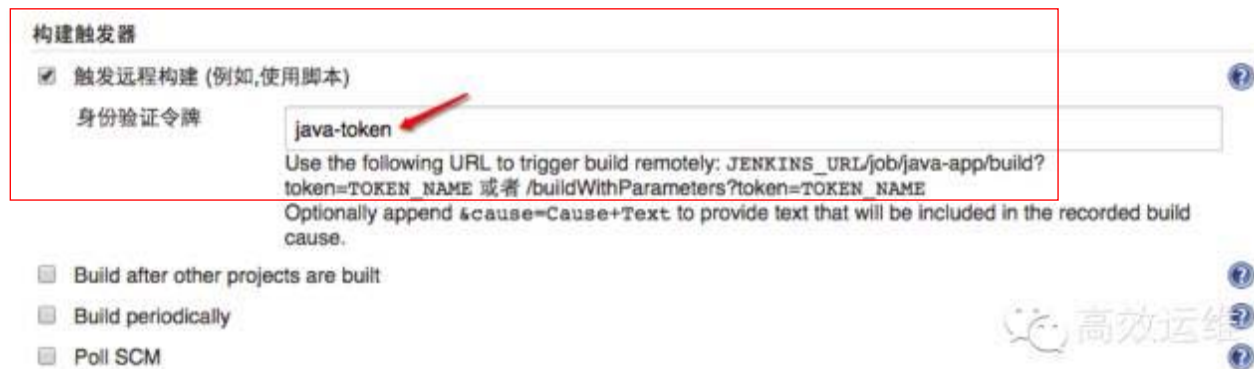
3.1 Jenkins配置Git源

Jenkins中新建项目java-app，并配置从Git拉取程序代码。具体如下：



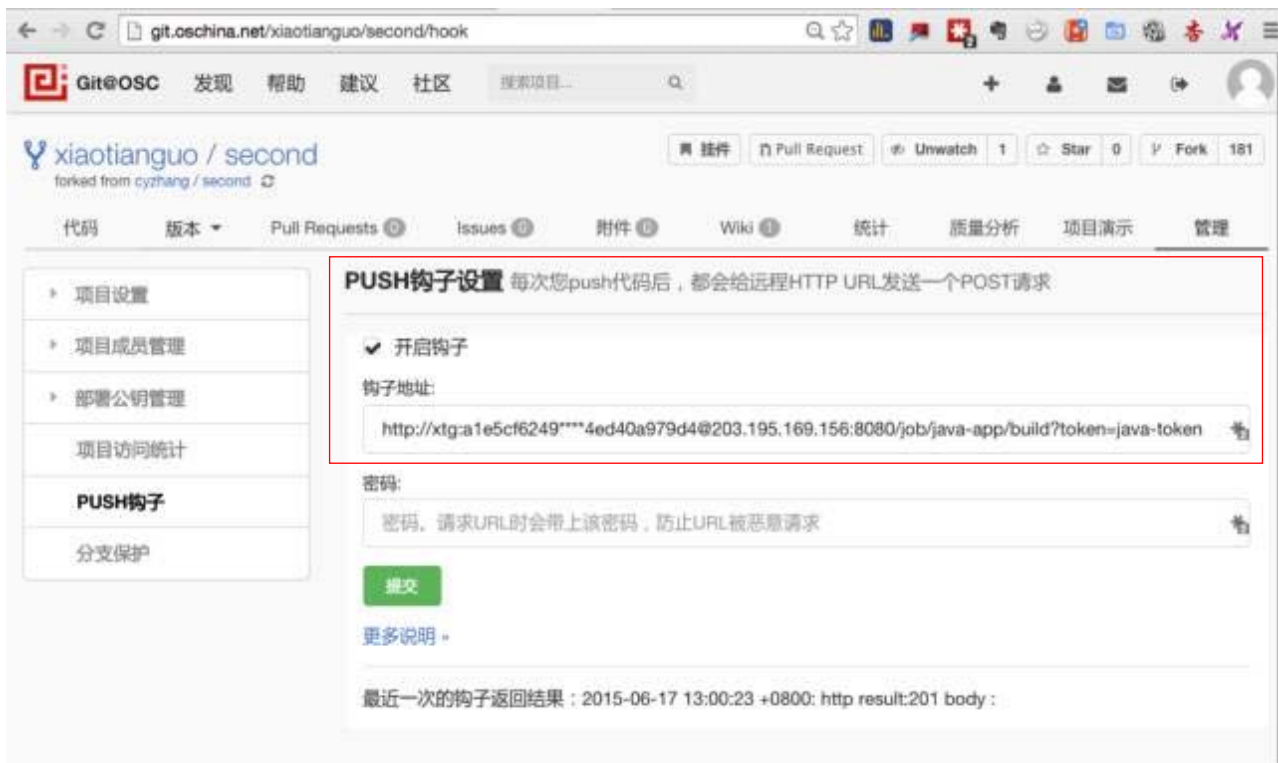
3.2 Jenkins配置远程构建

Jenkins中配置token，以供git远程调用时使用。



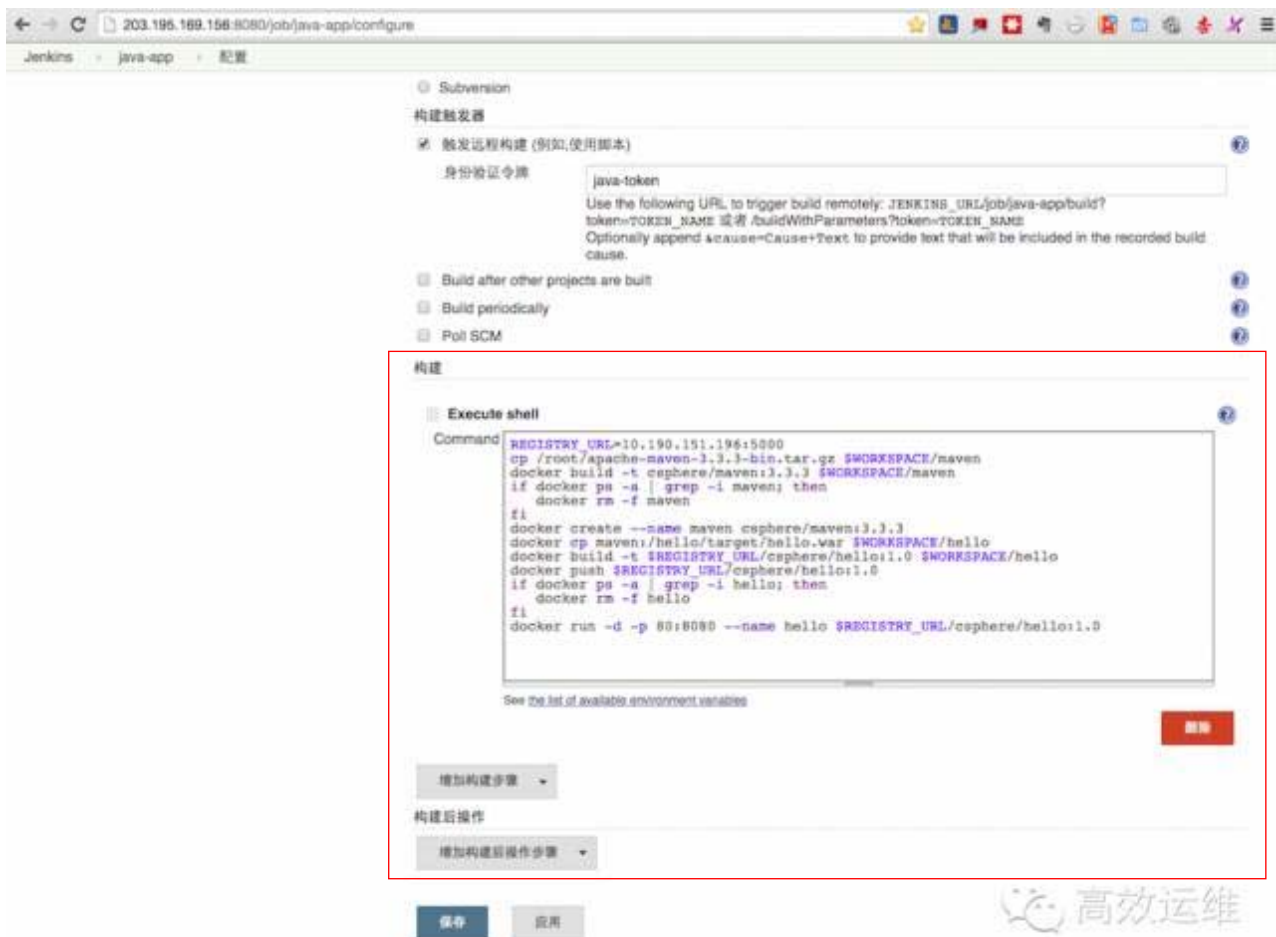
3.3 Git开启钩子

怎么让Git在接收到用户更新的代码后，把消息和任务传递给Jenkins呢？这借助于Git的hook功能，配置起来也非常简单，如下。



4. 配置Jenkins自动更新代码

Jenkins的主要工作是配置“远程构建”。在接收到Git传递过来的消息后，触发这个远程构建（到目标服务器），按照预定义的任务列表，执行一系列的工作，重建容器等。详见如下：



我们把其中最关键的Shell脚本内容摘抄出来。这些Docker相关操作，在第1部分“技术思路”已经提及，不再赘述。

```

REGISTRY_URL=10.190.151.196:5000
cp /root/apache-maven-3.3.3-bin.tar.gz $WORKSPACE/maven
docker build -t csphere/maven:3.3.3 $WORKSPACE/maven
if docker ps -a | grep -i maven; then
  docker rm -f maven
fi
docker create --name maven csphere/maven:3.3.3
docker cp maven:/hello/target/hello.war $WORKSPACE/hello
docker build -t $REGISTRY_URL/csphere/hello:1.0 $WORKSPACE/hello
docker push $REGISTRY_URL/csphere/hello:1.0
if docker ps -a | grep -i hello; then
  docker rm -f hello
fi
docker run -d -p 80:8080 --name hello $REGISTRY_URL/csphere/hello:1.0
  
```

高效运维

5. 效果图文详解

在2.3这个章节中，我们当时的操作如下，这个目的是向Git提交更新代码。


```
git add *
git commit -m "Modified the front-end display content"
git config --global user.email "xiaotianguo@foxmail.com"

git push
```

然后，输入git账号密码，然后等待神奇的一幕...  高效运维

当时并没有细说后续发生的事情，既然上面已经说清楚了原理，那我们就可以接下来说说实际发生的事情啦。

5.1 上传代码到Git

这里貌似整个过程已经完成并顺利退出。其实，后台的工作才刚刚开始哦。

```
Username for 'https://git.oschina.net': xiaotianguo@foxmail.com
Password for 'https://xiaotianguo@foxmail.com@git.oschina.net':
Counting objects: 22, done.
Compressing objects: 100% (7/7), done.
Writing objects: 100% (8/8), 596 bytes | 0 bytes/s, done.
Total 8 (delta 5), reused 0 (delta 0)
To https://git.oschina.net/xiaotianguo/second.git
   233d137..5cb5145  master -> master
root@VM-151-196-ubuntu:/home/ubuntu/second#
```

 高效运维

这时会触发Git服务器向相应的Jenkins服务器发出一个操作请求，此工作太过迅速，也没啥好说的，我们接下来看Jenkins都干啥子了。

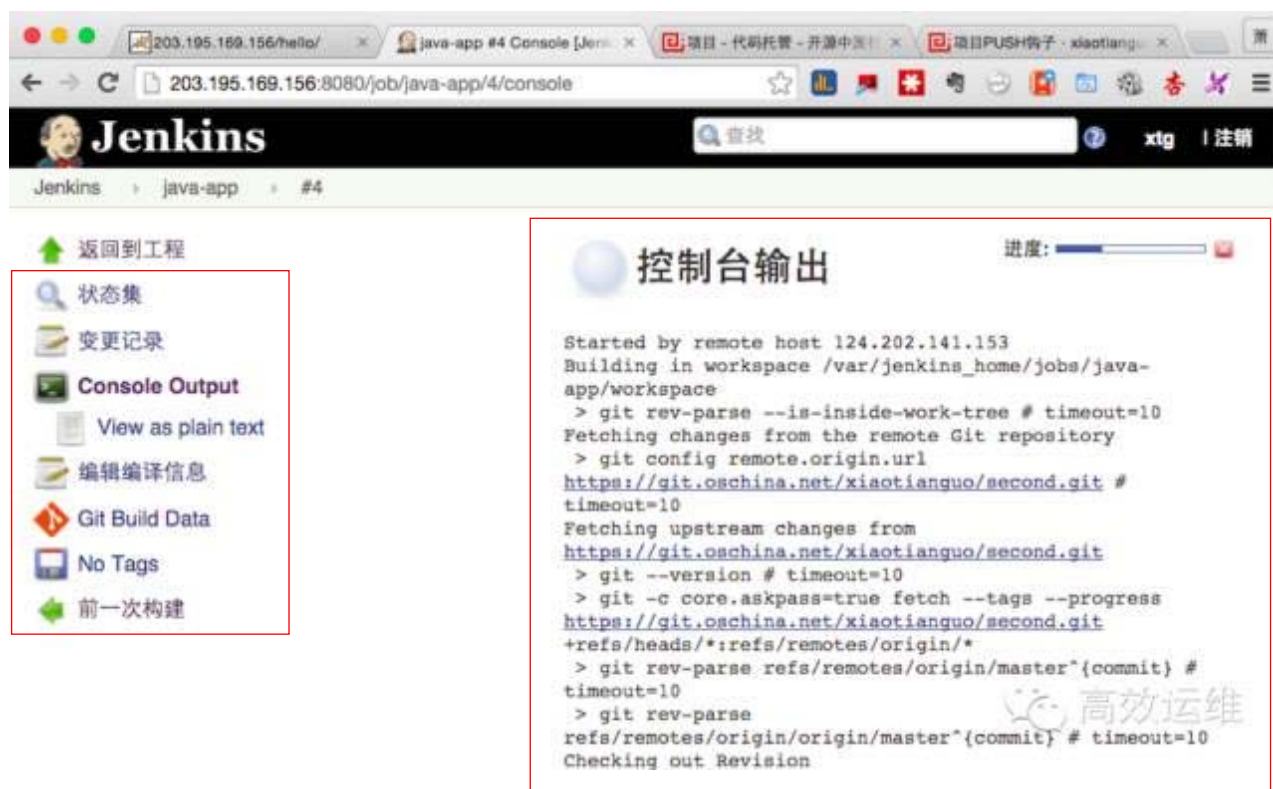
5.2 Jenkins进行的精彩互动

如下这个自动运转的过程，让我们有些许成就感，值得端杯咖啡（如果不喜欢茶的话），静静观赏。

1) Jenkins会自动“冒出来”一个构建任务。



2) 我们点进来，看看具体操作日志。是的，正在接受来自Git的任务。



3) 下载Maven相关的软件包（就是这个过程慢）。



```
Downloaded:
https://repo.maven.apache.org/maven2/org/apache/maven/mav
en-repository-metadata/2.2.1/maven-repository-metadata-
2.2.1.pom (2 KB at 1.6 KB/sec)
Downloading:
https://repo.maven.apache.org/maven2/org/apache/maven/mav
en-error-diagnostics/2.2.1/maven-error-diagnostics-
2.2.1.pom

Downloaded:
https://repo.maven.apache.org/maven2/org/apache/maven/mav
en-error-diagnostics/2.2.1/maven-error-diagnostics-
2.2.1.pom (2 KB at 0.8 KB/sec)
Downloading:
https://repo.maven.apache.org/maven2/org/apache/maven/mav
en-project/2.2.1/maven-project-2.2.1.pom

Downloaded:
https://repo.maven.apache.org/maven2/org/apache/maven/mav
en-project/2.2.1/maven-project-2.2.1.pom (3 KB at 0.9
KB/sec)
Downloading:
https://repo.maven.apache.org/maven2/org/apache/maven/mav
en-artifact-manager/2.2.1/maven-artifact-manager-
2.2.1.pom
```

4) 下载完成后，就开始利用 maven BUILD 新的 hello 项目包。



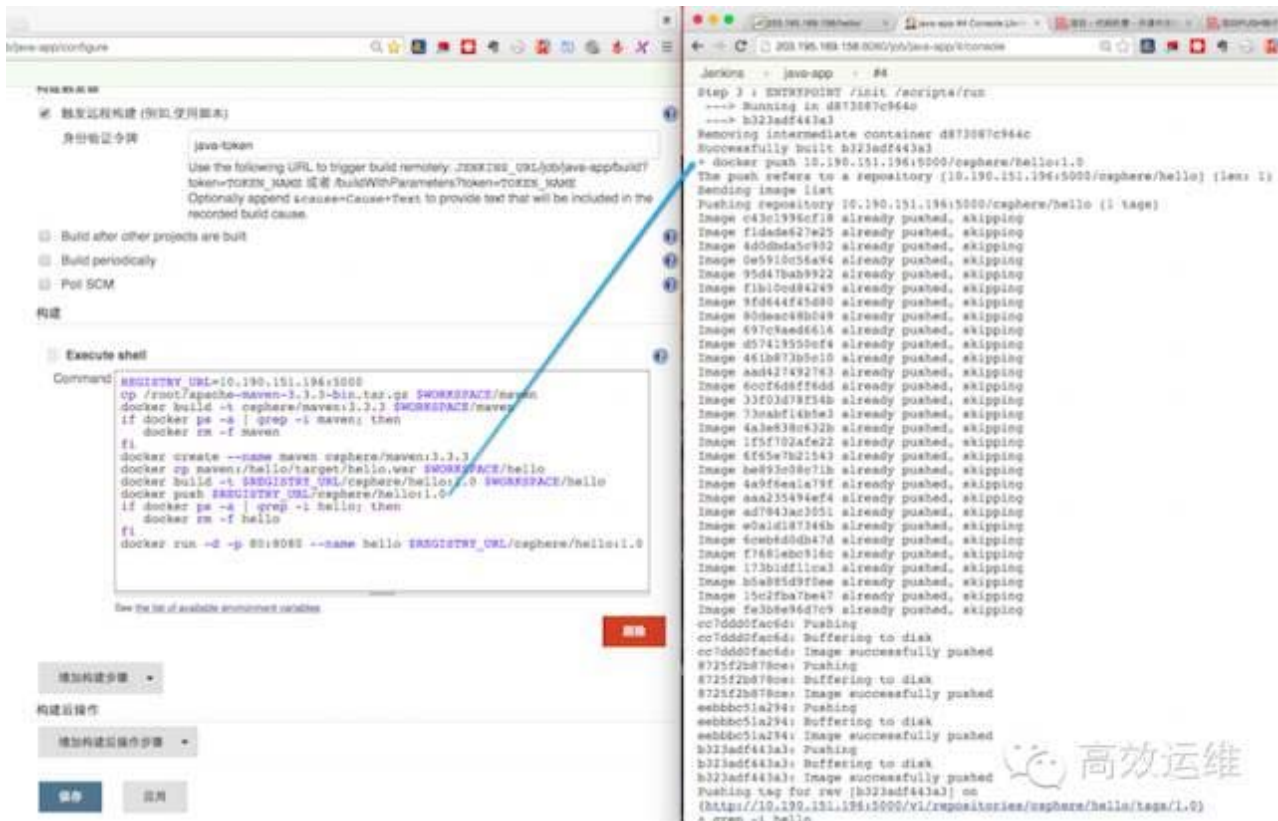
```
Downloaded:
https://repo.maven.apache.org/maven2/org/codehaus/plexus/
plexus-digest/1.0/plexus-digest-1.0.jar (12 KB at 10.5
KB/sec)

Downloaded:
https://repo.maven.apache.org/maven2/classworlds/classwor
lds/1.1-alpha-2/classworlds-1.1-alpha-2.jar (37 KB at
32.9 KB/sec)

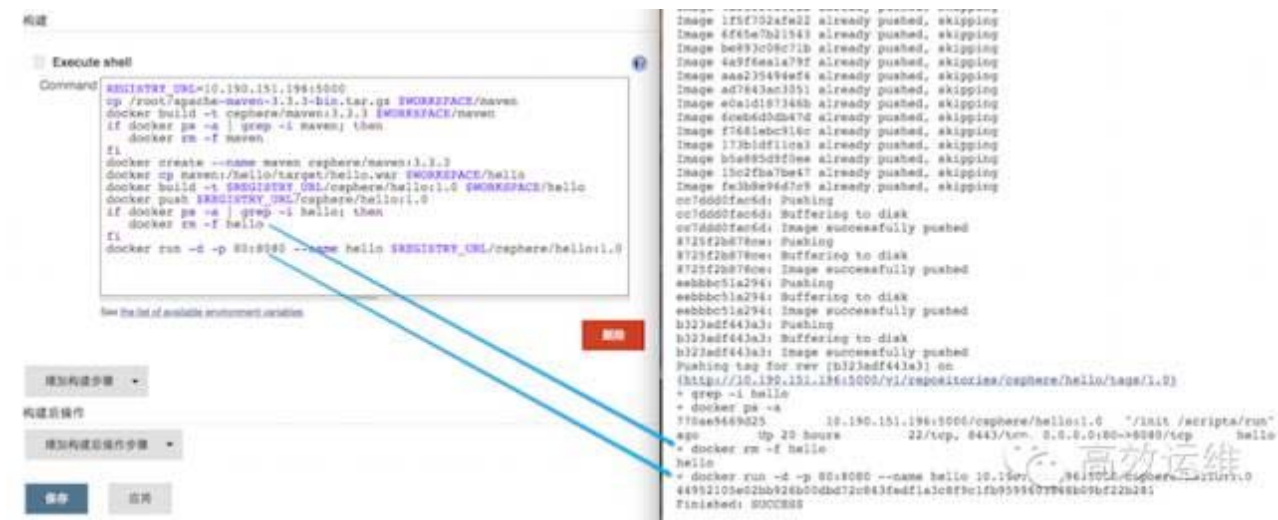
Downloaded:
https://repo.maven.apache.org/maven2/org/codehaus/plexus/
plexus-utils/3.0.5/plexus-utils-3.0.5.jar (226 KB at
195.5 KB/sec)

[INFO] Installing /hello/target/hello.war to
/root/.m2/repository/csphere/xc/0.0.1-SNAPSHOT/xc-0.0.1-
SNAPSHOT.war
[INFO] Installing /hello/pom.xml to
/root/.m2/repository/csphere/xc/0.0.1-SNAPSHOT/xc-0.0.1-
SNAPSHOT.pom
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 04:59 min
[INFO] Finished at: 2015-06-19T14:31:21+08:00
[INFO] Final Memory: 15M/40M
[INFO] -----
---> 4f0fe784f009
```

5) 然后重建 Maven 容器，构建新的 Image 并 Push 到 Docker 私有库中。



6) 最后，重新把Docker容器拉起来。这样，又新生了。呵呵



6. FAQ

问题1：采用这么相对复杂的办法（而不是把更新代码放在宿主机然后卷组映射），是因为项目基于JAVA么；是否PHP项目就可以采用更新代码放在宿主机然后卷组映射这种方式？

回答1：将代码拆分出容器，违背了集装箱原则。导致装卸复杂度增加。从货运工人角度考虑，整体才是最经济的。一切版本化。抛弃过去的文件分发。这是正途。至于文

件大小，大的war包也就50M或100M，在现有网络下不成问题，性能问题最好优化。另外建议关注docker 2 docker，p2p传输。

问题2：如果整体代码超过500m或者1g以上，整体集装箱是否就不太好了？如果容器与代码分离，镜像就100m左右（2层，base+服务），然后代码的话，是放到共享存储里，每个代码有更新，比如svn的代码，可以直接在共享存储里进行svn update就可以控制版本

回答2：如果你的代码500M，那只能说明业务开发该打板子了。

问题3：如果测试环境使用您提供的完整集装箱服务还行，但在生产环境，集群里运行docker做应用，如果每个容器都是有完整的代码，是否有点臃肿，不如每个集群节点里就运行基础服务镜像，通过卷组功能绑定共享存储里的代码，加上Crontab、Python和Shell脚本，这样每次代码更新就1次就行了。

回答3：环境一致性，在过去从来没有解决好。10年前我们做paas时，和这个做法类似。不是说不好，时代变了，用脚本东拼西凑，终究难有好的系统。不能只考虑现在的方便，容器技术和vm如果类比，我觉得会让自己下决定时很纠结。

补充3：脚本一般是典型的运维工程师思维，quick & dirty。一般很难做成一个产品或者系统。整体考虑和扩展性考虑都比较少。现在做docker的难点在于到底怎么看待它。到底是拿它做调度的基本单位，还是部署的基本单位？考虑清楚，再聊方案。

备注：上述问题的回答，主要由王利俊@cSphere和陈尔冬@华为完成。

关于作者

萧田国，男，硕士毕业于北京科技大学，触控科技运维负责人。拥有十多年运维及团队管理经验。先后就职于联想集团、搜狐畅游、智明星通及世纪互联等。曾经的云计算行业从业者，现在喜欢琢磨云计算及评测、云端数据库，及新技术在运维中的应用。主张管理学科和运维体系的融合、人性化运维管理，打造高效、专业运维团队。

张春源，目前任职cSphere。国内最早期的Docker实践者，在生产环境拥有一年多的Docker容器管理经验。深刻理解Docker对于开发、测试以及运维的价值。擅长利用Docker构建整个DevOps自动化平台。热爱专研Dockerfile这门艺术，并对CoreOS有深入研究。