Docker+Maven+Jenkins在Devops中完整应用

过去与现在

很早之前，当我们需要一个部署环境的时候，我们可能指的是一台PowerEdge R710 2U服务器，走一系列冗长的申请流程，然后上架到机房、调试网络、安装系统、调试环境、最终部署应用，就这样过去了**几个月**。
接着出现了虚拟化技术，我们在一台内部服务器使用Citrix XenApp划分出几台虚拟机，搭建了内部需求管理系统、SVN、测试环境等等，当需要新的机器时，我们只需要再次复制出一台虚拟机即可，现在只需要**几个小时**我们就能整理好一个环境。
到了2013年，docker出现了。当我需要搭建一个Jenkins环境时，我只需要执行两个命令：
docker pull jenkins/jenkins
docker run jenkins/jenkins
就可以开始使用Jenkins了，只花了我**几分钟**时间。
docker究竟是什么，我们应该如何在软件生产过程中使用呢?

docker基本概念

我们不打算深入的介绍docker的基础原理，只打算从最基本的应用的场景来说明我们必须理解的部分。
一个完整的Docker有以下几个部分组成：

- Docker **Client**【客户端】
- Docker Client是Docker架构中用户与Docker Daemon建立通信的客户端。
- Docker **Daemon**【守护进程】
- Daemon是Docker的守护进程，Docker Client通过命令行与Docker Damon通信，完成Docker相关操作。
- Docker **Image**【镜像】
- 包含了用户定义的应用和其相关的依赖，可以理解成是一张“系统盘”。
- Docker **Container**【容器】
- 镜像实例化之后就是容器，容器生成之后任何变化与镜像无关，可以理解成是“系统盘”安装的“系统”。

docker使用场景

我们假设的场景流程是这样：
现在我们有一个应用，托管在Git仓库上。管理层要求能够完整支持自动化的devops流程，所以我们需要能够使用CI服务器从Git仓库上拉取代码，编译打包之后生成镜像，上传至公司的私有仓库。最后能够使用CI服务器部署至应用服务器并启动。
进阶场景：
最后一步改为使用容器编排软件，拉取镜像并且部署。并且支持灰度发布，自动扩容（模拟场景）。
预设的环境
应用：SpringBoot + Maven
应用服务器环境：CentOS
Git仓库：Github
CI服务器：Jenkins
私有仓库：阿里云容器镜像服务
容器编排软件：Kubernetes

实施过程

在实施过程中，我们可以不用太关心具体SpringBoot的代码，只是用于说明Dockerfile如何构建。所以我们忽略编写应用，上传Github的过程，假设目前已经有已经上传好的应用，Jenkins也启动完毕，我们从这里讲起。

pom.xml编写

pom.xml中最重要的部分就是根据Dockerfile生成镜像的插件，下面给出关键部分的节点

```
 <properties>
  <java.version>1.8</java.version>
  <spring-cloud.version>Greenwich.SR2</spring-cloud.version>
  <docker.image.prefix>software5000</docker.image.prefix>
  <docker.repository>registry.cn-hangzhou.aliyuncs.com</docker.repository>
 </properties>

 <build>
  <plugins>
   <plugin>
    <groupId>com.spotify</groupId>
    <artifactId>dockerfile-maven-plugin</artifactId>
    <version>1.4.10</version>
    <executions>
     <execution>
      <id>default</id>
      <goals>
       <goal>build</goal>
       <goal>push</goal>
      </goals>
     </execution>
    </executions>
    <configuration>
     <repository>${docker.repository}/${docker.image.prefix}/$
{project.artifactId}</repository>
     <tag>${project.version}</tag>
     <buildArgs>
      <JAR_FILE>target/${project.build.finalName}.jar</JAR_FILE>
     </buildArgs>
    </configuration>
   </plugin>
  </plugins>
 </build>
```

镜像的打包实际上是由com.spotify.dockerfile-maven-plugin执行，重点在**configuration**标签下的几个子标签，稍后会好好解释。

Dockerfile

Dockerfile是最基础的docker 镜像描述文件，可以直接访问官方说明。

```
FROM openjdk:8u181-jdk-alpine
ARG workdir=/app
```

```
VOLUME ${workdir}
WORKDIR ${workdir}
ARG JAR_FILE
COPY ${JAR_FILE} app.jar
EXPOSE 8081
ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","app.jar"]
```

我们就上面的进行一下说明

FROM：是指从什么基本镜像继承增加自定义配置。这是大部分Dockerfile的第一行代码，因为Docker的推荐最佳实践中就是强调复用，复用社区或者他人已经提供的基础镜像。我们使用的是openjdk 8的基础镜像

ARG：是指Dockerfile中的参数变量，这里就可以看到上面*pom.xml*中有一个**buildArgs**标签，实际上就是用来在初始化Dockerfile时提供的参数。

VOLUME：是指Docker容器运行时的挂载点，用于方便容器和宿主机器之间磁盘访问，类似**共享文件夹**。

WORKDIR：指定了Docker容器运行时的基础根目录。默认命令会从指定的目录开始执行。

COPY/ADD：这个是构建镜像时的**核心命令**了，会将当前宿主机器中的文件拷贝到镜像当中，比如本例中就是将打包好的jar复制到镜像中。区别是ADD会自动解压（比如war就可以解压出来了），COPY不会。

EXPOSE：这个比较好理解，就是容器运行时，可以对外映射的端口。（当同一个容器在同一个宿主机器运行时，可以指定多个外部访问端口）

CMD/ENTRYPOINT：这同样也是**核心命令**，作用就是在启动时自动执行的命令。但是差异如下：

- CMD命令设置容器启动后默认执行的命令及其参数，但CMD设置的命令能够被docker run命令后面的命令行参数替换
- ENTRYPOINT配置容器启动时的执行命令（不会被忽略，一定会被执行，即使运行 docker run时指定了其他命令）

镜像仓库

镜像仓库使用了阿里云的容器镜像服务

我们在*pom.xml*中的属性中有两个变量：docker.repository 和 docker.image.prefix，如果你使用的是阿里云，那基本docker.repository固定就是registry.cn-hangzhou.aliyuncs.com，docker.image.prefix就是你创建的命名空间了。

接下来是要创建镜像仓库。我一开始理解的镜像仓库有错，我以为所有的镜像都是放在某一个镜像仓库下，实际上镜像仓库是指**某一个镜像多个版本的仓库**。所以针对我们的每个项目都要创建一个镜像仓库。

Jenkins推送仓库和拉取部署

最终串联的场景是这样，当版本已经测试完毕，测试判定可以执行正式发布动作，点击Jenkins中release任务的build按钮。而服务器则定时凌晨2点定时删除本地镜像并且拉取最新的镜像并且启动容器。
镜像版本控制，也就是docker的tag。我们在dockerfile:build的时候，给镜像打的版本是什么，如果不标记默认就是latest。这样推到仓库之后再拉取整个过程只要不标记版本号即可。
根据我们之前的pom文件，Jenkins中这样配置就会推送至阿里云的镜像仓库。

服务器定时任务执行的shell脚本，就是停止容器、删除容器、删除镜像。

```
# 停止容器
docker stop eureka-server
# 删除容器
docker rm eureka-server
# 删除镜像
docker rmi eureka-server
# 拉取镜像
docker pull registry.cn-hangzhou.aliyuncs.com/software5000/eureka-server

# 运行镜像
docker run -d --name=eureka-server -p 9001:8081 registry.cn-hangzhou.aliyuncs.com/software5000/eureka-server
```

总结

这篇文章主要是将docker，maven，jenkins整合，模拟实际工作的场景将所有流程串联在一起，并且使用里阿里云的仓库。但是在真正发布的时候，还需要考虑CI/CD，灰度发布等情况。我们并没有在这里完全解决掉。因为每个项目的环境和架构都不相同，所以还是要根据实际具体的情况来推进Devops。现在也正在研究Kubernetes的使用，现在能够确认的有一点。Kubernetes相当于接管了整个环境，所以我们的服务器都是需要安装Kubernetes的客户端，所有的应用也都变成容器化的运行方式。所以当环境切换到docker之后，再切换至Kubernetes会更简单，后续的维护也更加轻松。
另外，如果本文还是看得不是很明白可以先了解一下docker的基本命令和使用方法，自己建个虚拟机体验一下就能明白很多事情。