# 业务系统容器化迁移（For Developer）

Edit by JC

1. 开发/测试环境搭建：（已安装可以直接跳过）
2. 安装docker

详细步骤见同文档同目录《安装Docker.txt》

2） 安装https证书：

在docker所在的主机上新建目录:

mkdir -p /etc/docker/certs.d/harbor-master.yooli-me.com

然后将访问私有远程仓库的https证书加入到该目录中。（文档同目录的ca.crt文件）

1. 拉取基础镜像并重打tag：

登录Harbor测试环境，地址：<https://harbor-master.yooli-me.com>

账号：deploy\_test 密码：LoFkO73THnV9byQC

进入“baseimages”工程拉取指定镜像，例如：

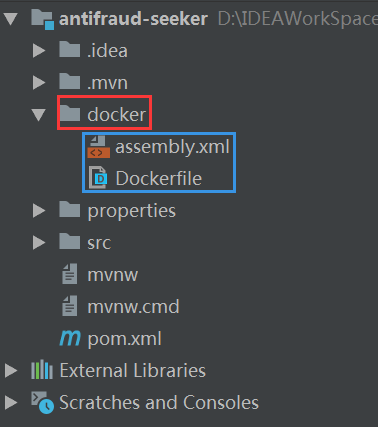
docker pull harbor-master.yooli-me.com/baseimages/orajdk8-centos7:v1.1

docker tag harbor-master.yooli-me.com/baseimages/orajdk8-centos7:v1.1 orajdk8-centos7:v1.1

1. 项目迁移（重点关注）

前言：为了便于开发和测试镜像，这里提供两种插件供开发者使用，分别是：fabric8的docker-maven-plugin和docker integration。前者是maven插件，通过在pom.xml中引入相关配置即可使用，适用于在docker环境中快速构建、删除、推送镜像、启动容器等操作；后者是IDEA集成的工具插件，需要手动安装并重启生效，它更适用于实时查看docker环境的镜像详情、容器详情、删除镜像以及快速启动容器等操作。（两种插件功能有重合亦有互补，建议开发者两种均学习使用）

**1）在项目根模块路径下新建docker文件夹，并将对应项目类型的Dockerfile和 assembly.xml拷贝进去**



**2）在项目主pom.xml中引入配置：（注：由于运行环境的区别，SpringBoot和 SpringMVC需按各自指定方式配置；实际开发中请勿直接copy以下配置信息， 此配置信息只作文档说明使用，开发者可在文档同目录下获取指定配置文件）**

2.1）SpringBoot项目迁移（依赖JDK环境）：

①、更新父工程：

<parent>  
 <groupId>tech.yooli.maven.dependencies</groupId>  
 <artifactId>tech-yooli-maven-dependencies</artifactId>  
 <version>0.6.0.RELEASE</version>  
</parent>

②引入项目相关变量配置：

<properties>  
 <!--项目组名称-->  
 <project.group>settle</project.group>  
 <!--多模块配置为欲打成镜像的jar包的子模块名称，单模块直接配置为“.”，默认情况为单模块-->  
 <serverModuleRootPath>settle-check-web</serverModuleRootPath>  
 <!--需要构建为镜像的jar包名称-->  
 <serverModuleName>settle-check-web</serverModuleName>  
 <!--服务发布的名称，必须和日志路径名完全一致，且必须是小写字母-->  
 <server.publish.name>settle-check</server.publish.name>  
 <!--服务需要暴露的端口号-->  
 <bind.server.port>8100</bind.server.port>  
</properties>

③如果是多模块项目，需要在每个子模块中引入配置（单模块项目直接跳过）：

<plugin>  
 <groupId>io.fabric8</groupId>  
 <artifactId>docker-maven-plugin</artifactId>  
 <configuration>  
 <skip>true</skip>  
 <skipBuild>true</skipBuild>  
 <skipPush>true</skipPush>  
 <skipSave>true</skipSave>  
 </configuration>  
</plugin>

2.2）SpringMVC项目迁移（依赖tomcat环境）：

①引入项目相关变量配置：

<properties>  
 <!--docker插件定义参数start-->  
<docker.image.repositories>harbor-master.yooli-me.com</docker.image.repositories>  
 <harbor.account>deploy\_test</harbor.account>  
 <harbor.passwd>LoFkO73THnV9byQC</harbor.passwd>  
 <project.group>tmp</project.group>  
 <bind.server.port>8080</bind.server.port>  
 <serverModuleName>loanweb</serverModuleName>  
 <serverModuleRootPath>.</serverModuleRootPath>  
 <server.publish.name>loanweb</server.publish.name>  
 <local.docker.host>172.16.4.147:2377</local.docker.host>  
 <healthCheck4Dev>http://ipAddress:${bind.server.port}/</healthCheck4Dev>  
 <log.root.volume>/data/home/www/yooli/logs</log.root.volume>  
 <localDockerFileDir>${project.basedir}/docker</localDockerFileDir>  
 <assemblyFile>${project.basedir}/docker/assembly.xml</assemblyFile> <serverJar>maven/${serverModuleRootPath}/target/${serverModuleName}.war</serverJar>  
 <profile.branch>none</profile.branch>  
 <!--docker插件定义参数end-->  
</properties>

②引入docker插件：

<build>  
 <pluginManagement>  
 <plugins>  
 <plugin>  
 <groupId>io.fabric8</groupId>  
 <artifactId>docker-maven-plugin</artifactId>  
 <version>0.30.0</version>  
 <inherited>true</inherited>  
 </plugin>  
 </plugins>  
 </pluginManagement>  
 <plugins>  
 <!--获取时间戳-->  
 <plugin>  
 <groupId>org.codehaus.mojo</groupId>  
 <artifactId>buildnumber-maven-plugin</artifactId>  
 <version>1.4</version>  
 <configuration>  
 <timestampFormat>yyyyMMdd\_HHmmss</timestampFormat>  
 </configuration>  
 <executions>  
 <execution>  
 <goals>  
 <goal>create-timestamp</goal>  
 </goals>  
 </execution>  
 </executions>  
 <inherited>false</inherited>  
 </plugin>  
 <!--获取时间戳-->  
 <!--Docker maven plugin-->  
 <plugin>  
 <groupId>io.fabric8</groupId>  
 <artifactId>docker-maven-plugin</artifactId>  
 <configuration>  
 <!--docker主机地址-->  
 <dockerHost>tcp://${local.docker.host}</dockerHost>  
 <!--远程仓库地址-->  
 <registry>${docker.image.repositories}</registry>  
 <!--停止后是否保留容器-->  
 <keepContainer>false</keepContainer>  
 <!--打印构建过程-->  
 <verbose>true</verbose>  
 <!--权限验证-->  
 <authConfig>  
 <push>  
 <username>${harbor.account}</username>  
 <password>${harbor.passwd}</password>  
 </push>  
 </authConfig>  
 <images>  
 <image>  
 <!--镜像别名，用于启动容器时引用-->  
 <alias>${server.publish.name}-v${project.version}-${timestamp}</alias>  
 <!--镜像名称:tag-->  
 <name>${docker.image.repositories}/${project.group}/${server.publish.name}-${profile.branch}:v${project.version}-${timestamp}</name>  
 <build>  
 <!--不使用maven缓存-->  
 <noCache>true</noCache>  
 <!--指定dockerfile位置-->  
 <dockerFileDir>${localDockerFileDir}</dockerFileDir>  
 <!--如果是多模块项目且需要在根目录下面打包，则需要在这里定义assembly-->  
 <assembly>  
 <descriptor>${assemblyFile}</descriptor>  
 </assembly>  
 <!--为镜像添加一个latest标签-->  
 <tags><tag>latest</tag></tags>  
 <!--向dockerfile传参-->  
 <args>  
 <JAR\_FILE>${serverJar}</JAR\_FILE>  
 <EXPOSE\_PORT>${bind.server.port}</EXPOSE\_PORT>  
 <LOG\_ROOT\_VOLUME>${log.root.volume}</LOG\_ROOT\_VOLUME>  
 <PROJECT\_ARTIFACTID>${serverModuleName}</PROJECT\_ARTIFACTID>  
 <SERVER\_PUBLISH\_NAME>${server.publish.name}</SERVER\_PUBLISH\_NAME>  
 </args>  
 </build>  
 <run>  
 <!--该参数在0.29.0版本以后才被加入-->  
 <autoRemove>true</autoRemove>  
 <!--容器名称，引用上面的别名-->  
 <namingStrategy>alias</namingStrategy>  
 <ports>  
 <port>${bind.server.port}:${bind.server.port}</port>  
 <!--远程debug的端口，一般不需要改动-->  
 <port>7474:7474</port>  
 </ports>  
 <volumes>  
 <bind>  
 <!--日志本地存储-->  
 <volume>${log.root.volume}/${server.publish.name}:${log.root.volume}/${server.publish.name}</volume>  
 </bind>  
 </volumes>  
 <wait>  
 <http>  
 <url>${healthCheck4Dev}</url>  
 </http>  
 <time>10000</time>  
 </wait>  
 <log>  
 <prefix>Container @ ${project.name}-v${project.version}-${timestamp} log：</prefix>  
 <date>default</date>  
 </log>  
 </run>  
 </image>  
 </images>  
 </configuration>  
 <dependencies>  
 <dependency>  
 <groupId>commons-codec</groupId>  
 <artifactId>commons-codec</artifactId>  
 <version>1.11</version>  
 </dependency>  
 </dependencies>  
 </plugin>  
 <!--Docker maven plugin-->  
 </plugins>  
</build>

如果是多模块项目，需要在每个子模块中引入配置（单模块项目直接跳过）：

<build>  
 <plugins>  
 <plugin>  
 <groupId>io.fabric8</groupId>  
 <artifactId>docker-maven-plugin</artifactId>  
 <configuration>  
 <skip>true</skip>  
 <skipBuild>true</skipBuild>  
 <skipPush>true</skipPush>  
 <skipSave>true</skipSave>  
 </configuration>  
 </plugin>  
 </plugins>  
</build>

**3）构建镜像命令：**

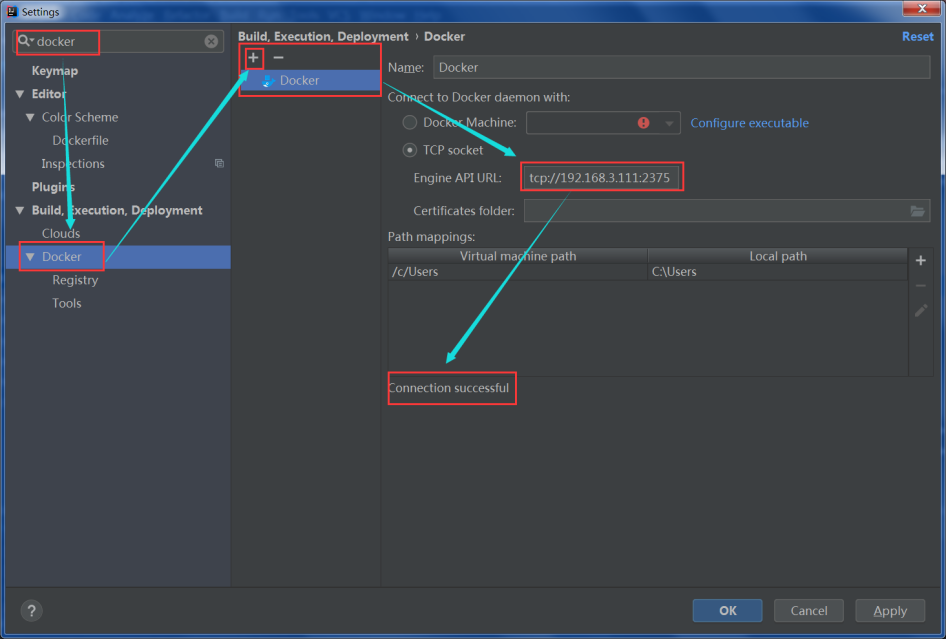
详见文档同目录cmd.txt

1. docker Integration插件使用：（建议使用）

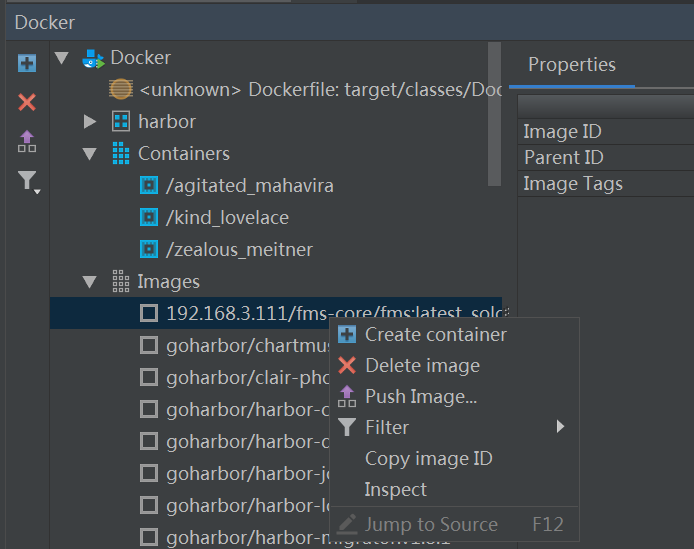
1) 插件安装：

打开IDEA->File->Setting->Plugins，搜索docker，然后点击安装docker integration插件，安装完后重启IDEA生效。

2) 配置插件：



3) 可以通过该插件实现查看镜像和容器信息、快速启动容器、删除容器、推 送镜像，其他功能请安装后自行摸索：



1. Dockerfile常用指令：（了解即可）

前言：同文档目录中已经提供了两份Dockerfile模板（SpringBoot/SpringMVC），但这并不能适用所有实际项目。这里我们还将列出实际开发中需要用到的一些常用Dockerfile指令，它们是开发人员需要简单了解的内容。值得注意的是，Docker镜像文件使用的是分层构建、联合加载的overlay2存储类型，每一行指令都会使得镜像文件在原有镜像上增加一层。因此，除非必要信息，请惜字如金！

1. **FROM 用于指定欲制作镜像的基础镜像，且必须是第一条有效指令**

**用法：**

**FROM <image>**

**FROM <image>:<tag>**

**FROM <image>:<digest>**

1. **RUN 用于执行shell指令(构建容器时执行)**

**用法：**

**RUN <command>**

**RUN ["executable", "param1", "param2"]**

1. **CMD 用于指定容器启动时要执行的命令(容器启动时执行)**

**用法基本同上**

1. **LABEL 标注作用，无实际功能意义，格式为key=value**

**用法：**

**LABEL key1="value1" \**

**key2="value2"**

1. **MAINTAINER 用于指定镜像的开发者与维护者信息，建议写上**

**用法：**

**MAINTAINER "zhangsan san.zhang@yooli.com"**

1. **EXPOSE 服务暴露的端口，只起到声明作用，不做端口映射**

**用法：**

**EXPOSE 8080**

1. **ENV 用于指定环境变量**

**用法：**

**ENV key value**

1. **ADD 用于将文件拷贝至容器中,支持源地址为远程url**

**用法：**

**ADD src dest**

1. **COPY 用于拷贝本地文件**

**用法基本同上**

1. **ENTRYPOINT 用于指定容器启动入口**

**用法：**

**ENTRYPOINT ["executable", "param1", "param2"]**

1. **VOLUME 用于指定存储卷，用于持久化保存数据使用**

**用法：**

**VOLUME ["/var/log/"]**

**VOLUME /var/log**

1. **USER 用于指定启动容器的用户**

**用法：**

**USER daemo**

**USER UID**

1. **WORKDIR 用于指定工作目录**

**用法：**

**WORKDIR /date/test**

1. **ARG 用于定义了一个变量，用于创建镜像时从通过--build-arg key=value向容器中传递值，运 行时无效**

**用法：**

**ARG key**

**ARG key=默认值**

1. **ONBUILD 当当前镜像被当做基础镜像制作成另一个镜像时，该指令指定的命令会被执行**

**用法：**

**ONBUILD RUN ls -a**

1. **STOPSIGNAL 当容器退出时发出信号**

**用法：**

**STOPSIGNAL signal**

五、FAQ

1、SpringBoot升级父工程出现依赖缺失和编译报错怎么办？

答：config\_SpringBoot中提供了两种SpringBoot项目容器化改造的方式，一个是继承架构组新发布的tech-yooli-maven-dependencies父工程，一个是直接拷贝maven插件的方式。如果你的项目通过升级父工程的方式改造出现编译异常或报错，可选择不继承父工程的方式进行升级。

2、容器化改造完以后会影响传统的通过jar包物理部署的方式吗？

答：构建镜像插件docker-maven-plugin并未绑定任何maven原生的任何打包命令， 构建镜像命令只有加上特有的docker:build才会驱动docker插件，因此不会影响传统的jar包物理部署方式

3、需要依赖第三方插件的服务如何迁移？

答：部分服务需要依赖第三方插件，迁移时就需要将服务依赖的插件构建到镜像中去，一般通过修改Dockerfile和Assembly.xml来实现添加。如果依赖的插件比较小，建议将插件加入到工程并提交到Git；如果依赖的插件较大或环境较为特殊的，可联系架构组提供与服务对应的基础镜像。

1. 基础镜像版本众多，该如何选择？

答：基础镜像由架构组维护，项目链接为：

<http://gitlab.yooli-in.com/YooliRDCenter/YooliAsset/Architecture/baseimages.git>

基础镜像每一个版本的升级都会伴有部分功能的增加，新版本完全向下兼容。如果需要了解各镜像升级的内容，可点击工程链接查看镜像readme.md介绍。

5、K8S上线流程：

