idea集成docker实现镜像打包一键部署

1、Docker开启远程访问

```
#修改该Docker服务文件
vi /lib/systemd/system/docker.service
#修改ExecStart这行
ExecStart=/usr/bin/dockerd  -H tcp://0.0.0.0:2375  -H unix:///var/run/docker.sock
```

> 将文件内的 ExecStart注释。 新增如上行。
>
> #ExecStart=/usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
> ExecStart=/usr/bin/dockerd -H tcp://0.0.0.0:2375-H unix:///var/run/docker.sock

```
#重新加载配置文件
systemctl daemon-reload
#重启服务
systemctl restart docker.service
#查看端口是否开启
netstat -nlpt     #如果找不到netstat命令，可进行安装。yum install net-tools
#直接curl看是否生效
curl http://127.0.0.1:2375/info
```

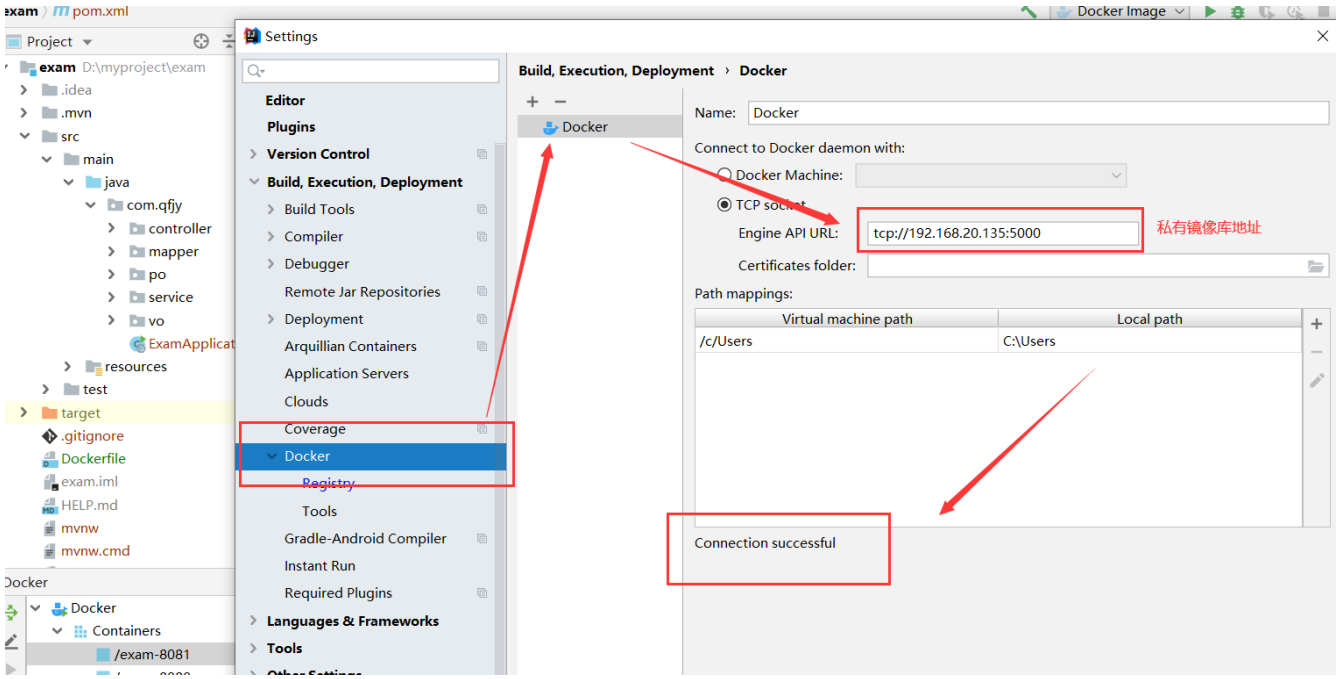
2、IDEA安装Docker插件

> 打开Idea，从File->Settings->Plugins->Install JetBrains plugin进入插件安装界面，
>
> 在搜索框中输入docker，可以看到Docker integration，点击右边的Install按钮进行安装。
>
> 安装后重启Idea。

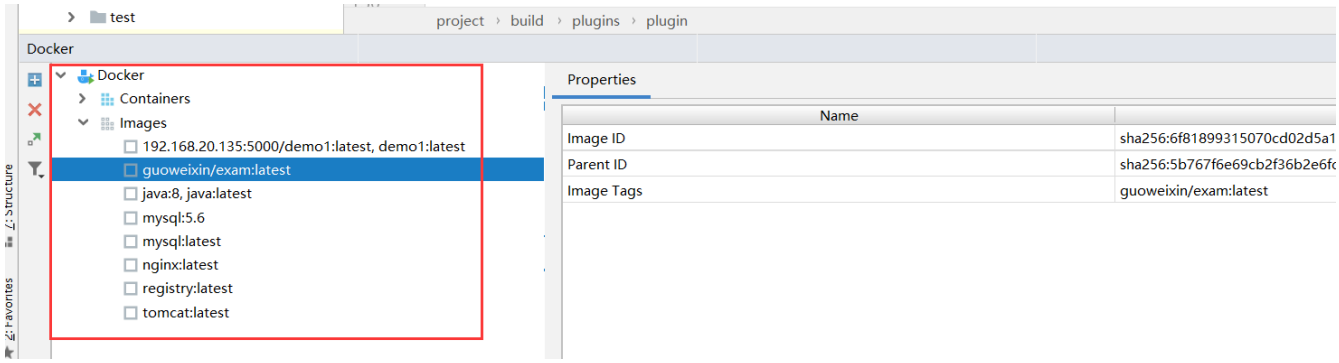
3、IDEA配置docker

配置docker，连接到远程docker服务。

从File->Settings->Build,Execution,Deployment->Docker打开配置界面



连接成功后，在IDEA工具中即可 操作Docker：



4、docker-maven-plugin

传统过程中，打包、部署、等。

而在持续集成过程中，项目工程一般使用 Maven 编译打包，然后生成镜像，通过镜像上线，能够大大提供上线效率，同时能够快速动态扩容，快速回滚，着实很方便。**docker-maven-plugin** 插件就是为了帮助我们在Maven工程中，通过**简单的配置，自动生成镜像并推送到仓库**中。

pom.xml

```
<properties>
     <docker.image.prefix>guoweixin</docker.image.prefix>
</properties>

 <build>
     <plugins>
         <plugin>
             <groupId>org.springframework.boot</groupId>
             <artifactId>spring-boot-maven-plugin</artifactId>
             <version>1.0.0</version>
```

```
        </plugin>

         <plugin>
               <groupId>com.spotify</groupId>
               <artifactId>docker-maven-plugin</artifactId>
               <version>1.0.0</version>

               <configuration>
                   <!-- 镜像名称  guoweixin/exam-->
                   <imageName>${docker.image.prefix}/${project.artifactId}</imageName>
                   <!--指定标签-->
                   <imageTags>
                       <imageTag>latest</imageTag>
                   </imageTags>
                   <!-- 基础镜像jdk 1.8-->
                   <baseImage>java</baseImage>
                   <!-- 制作者提供本人信息 -->
                   <maintainer>guoweixin guoweixin@aliyun.com</maintainer>
                   <!--切换到/ROOT目录 -->
                   <workdir>/ROOT</workdir>
                   <cmd>["java", "-version"]</cmd>
                   <entryPoint>["java", "-jar", "${project.build.finalName}.jar"]
</entryPoint>

                   <!-- 指定 Dockerfile 路径
                     <dockerDirectory>${project.basedir}/src/main/docker</dockerDirectory>
                  -->

                   <!--指定远程 docker api地址-->
                   <dockerHost>http://192.168.20.135:2375</dockerHost>

                   <!-- 这里是复制 jar 包到 docker 容器指定目录配置 -->
                   <resources>
                       <resource>
                           <targetPath>/ROOT</targetPath>
                           <!--用于指定需要复制的根目录，${project.build.directory}表示target目
录-->
                           <directory>${project.build.directory}</directory>
                           <!--用于指定需要复制的文件。${project.build.finalName}.jar指的是打包
后的jar包文件。-->
                           <include>${project.build.finalName}.jar</include>
                       </resource>
                   </resources>
               </configuration>

           </plugin>


     </plugins>
</build>
```

Dockerfile

如上用docker-maven插件 自动生成如下文件：

```
FROM java
MAINTAINER guoweixin guoweixin@aliyun.com
WORKDIR /ROOT
ADD /ROOT/qfnj-0.0.1-SNAPSHOT.jar /ROOT/
ENTRYPOINT ["java", "-jar", "qfnj-0.0.1-SNAPSHOT.jar"]
CMD ["java", "-version"]
```

5、执行命令

对项目进行 打包。并构建镜像 到Docker 上。

```
mvn clean package docker:build
```

6、IDEA 操作Docker

7、扩展配置

绑定Docker 命令到 Maven 各个阶段

> 我们可以绑定 Docker 命令到 Maven 各个阶段，
>
> 我们可以把 Docker 分为 build、tag、push，然后分别绑定 Maven 的 package、deploy 阶段，
>
> 我们只需要执行`mvn deploy`就可以完成整个 build、tag、push操作了，当我们执行`mvn build`就只完成 build、tag 操作。

```
<executions>
    <!--当执行mvn package 时，执行:  mvn clean package docker:build -->
    <execution>
        <id>build-image</id>
        <phase>package</phase>
        <goals>
            <goal>build</goal>
        </goals>
    </execution>
      <!--当执行mvn package 时，会对镜像进行 标签设定-->
     <execution>
         <id>tag-image</id>
         <phase>package</phase>
         <goals>
             <goal>tag</goal>
         </goals>
         <configuration>
             <image>${docker.image.prefix}/${project.artifactId}:latest</image>
```

```
<newName>docker.io/${docker.image.prefix}/${project.artifactId}:${project.version}
 </newName>
         </configuration>
    </execution>
    <execution>
        <id>push-image</id>
        <phase>deploy</phase>
        <goals>
            <goal>push</goal>
        </goals>
        <configuration>

 <imageName>docker.io/${docker.image.prefix}/${project.artifactId}:${project.version}
</imageName>
        </configuration>
    </execution>

</executions>
```

完整pom.xml如下：

```
<properties>
    <java.version>1.8</java.version>
    <!-- 镜像 前缀姓名-->
    <docker.image.prefix>guoweixin</docker.image.prefix>
</properties>

 <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        ..................
</dependencies>

<build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>

            <plugin>
                <groupId>com.spotify</groupId>
                <artifactId>docker-maven-plugin</artifactId>
                <version>1.0.0</version>

                <configuration>
                    <!-- 镜像名称  guoweixin/exam-->
                    <imageName>${docker.image.prefix}/${project.artifactId}</imageName>
                    <!--指定标签-->
                    <imageTags>
```

```
                        <imageTag>latest</imageTag>
                    </imageTags>
                    <!-- 基础镜像jdk 1.8-->
                    <baseImage>java</baseImage>
                    <!-- 制作者提供本人信息 -->
                    <maintainer>guoweixin guoweixin@aliyun.com</maintainer>
                    <!--切换到/ROOT目录 -->
                    <workdir>/ROOT</workdir>
                    <cmd>["java", "-version"]</cmd>
                    <entryPoint>["java", "-jar", "${project.build.finalName}.jar"]
</entryPoint>

                    <!-- 指定 Dockerfile 路径
                      <dockerDirectory>${project.basedir}/src/main/docker</dockerDirectory>
                    -->

                    <!--指定远程 docker api地址-->
                    <dockerHost>http://192.168.20.135:2375</dockerHost>

                    <!-- 这里是复制 jar 包到 docker 容器指定目录配置 -->
                    <resources>
                        <resource>
                            <targetPath>/ROOT</targetPath>
                            <!--用于指定需要复制的根目录，${project.build.directory}表示target目
录-->
                            <directory>${project.build.directory}</directory>
                            <!--用于指定需要复制的文件。${project.build.finalName}.jar指的是打包
后的jar包文件。-->
                            <include>${project.build.finalName}.jar</include>
                        </resource>
                    </resources>
                </configuration>

                <!--当执行mvn package 时，执行：  mvn clean package docker:build -->
                <executions>
                    <execution>
                        <id>build-image</id>
                        <phase>package</phase>
                        <goals>
                            <goal>build</goal>
                        </goals>
                    </execution>
                </executions>

            </plugin>

        </plugins>
    </build>
```

总结：

当我们执行`mvn package`时，执行 build、tag 操作，

当执行`mvn deploy`时，执行build、tag、push 操作。

如果我们想跳过 docker 某个过程时，只需要：

- `-DskipDockerBuild` 跳过 build 镜像
- `-DskipDockerTag` 跳过 tag 镜像
- `-DskipDockerPush` 跳过 push 镜像
- `-DskipDocker` 跳过整个阶段

例如：我们想执行 package 时，跳过 tag 过程，那么就需要`mvn package -DskipDockerTag`