# Docker集成文档

## 文档变更历史

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **主要更改内容** | **更改人** | **更改日期** |
| **1** | **初始文档** | **韩邦雪** | **2018-12-25** |

目录

[一 docker简介](#_Toc18390_WPSOffice_Level1) [1](#_Toc18390_WPSOffice_Level1)

[二 docker优势](#_Toc10302_WPSOffice_Level1) [1](#_Toc10302_WPSOffice_Level1)

[三 集成环境](#_Toc22845_WPSOffice_Level1) [2](#_Toc22845_WPSOffice_Level1)

[四 集成配置](#_Toc26738_WPSOffice_Level1) [3](#_Toc26738_WPSOffice_Level1)

[五 注意事项](#_Toc7825_WPSOffice_Level1) [4](#_Toc7825_WPSOffice_Level1)

# 一 docker简介

Docker 是一个开源的应用容器引擎，基于 Go 语言 并遵从Apache2.0协议开源。Docker 可以让开发者打包他们的应用以及依赖包到一个轻量级、可移植的容器中，然后发布到任何流行的 Linux 机器上，也可以实现虚拟化。容器是完全使用沙箱机制，相互之间不会有任何接口，更重要的是容器性能开销极低。

# 二 docker优势

## 1.简化配置

虚拟机的最大好处是能在你的硬件设施上运行各种配置不一样的平台(软件, 系统), Docker在降低额外开销的情况下提供了同样的功能. 它能让你将运行环境和配置放在代码汇总然后部署, 同一个Docker的配置可以在不同的环境环境中使用, 这样就降低了硬件要求和应用环境之间耦合度.

## 2.代码流水线管理

   代码从开发者的机器到最终在生产环境上的部署, 需要经过很多的中坚环境. 而每一个中间环境都有自己微小的差别,  Docker给应用提供了一个从开发到上线均一致的环境, 让代码的流水线变得简单不少.

## 3.提升开发效率

不同环境中, 开发者的共同目标:

一 , 想让开发环境尽量贴近生产环境.

二 , 想快速搭建开发环境

开发环境的机器通常内存比较小, 之前使用虚拟的时候, 我们经常需要为开发环境的机器加内存, 而现在Docker可以轻易的让几十个服务在Docker中跑起来.

## 4.隔离应用

开发时会在一个台机器上运行不同的应用.

一, 为了降低成本, 进行服务器整合

二, 将一个整体式的应用拆分成低耦合的单个服务(微服务架构)

## 5.整合服务器

Docker隔离应用的能力使得Docker可以整合多个服务器以降低成本. 由于没有多个操作系统的内存占用, 以及能在多个实例之间共享没有使用的内存, Docker可以比虚拟机提供更好的服务器整合解决方案.

## 6.调试能力

Docker提供了很多的工具, 这些工具不一定只是针对容器, 但是却适用于容器. 他们提供了很多功能, 包括可以为容器设置检查点, 设置版本, 查看两个容器之间的差别, 这些特性可以帮助调试Bug.

## 7.多租户环境

多租户环境的应用中, 它可以避免关键应用的重写.我们一个特别的关于这个场景的例子是为loT(物联网)的应用开发一个快速, 易用的多租户环境. 这种多租户的基本代码非常复杂, 很难处理, 重新规划以应用不但消耗时间, 也浪费金钱.

使用Docker, 可以为每一个租户的应用层的多个实例创建隔离的环境, 这不仅简单而且成本低廉, 因为Docker环境启动的速度快, diff命令很高效.

## 8.快速部署

Docker为进程创建一个容器, 不需要启动一个操作系统, 时间缩短为秒级别.

可以在数据中心创建销毁资源而无须担心重新启动带来的开销. 通常数据中心的资源利用率只有30% , 通过使用Docker并进行有效的资源分配可以提高资源的利用率.

# 三 集成环境

1. Springboot2.0+
2. Jdk1.8+
3. Tomcat8.0+

# 四 集成配置

1. 在项目根目录配置Dockerfile文件，这个文件用来生成docker镜像、配置jvm参数等。**第一次上线需要跟运维同学确认自己项目启动参数**。（Dockerfile文件我会在群里面发模板，模板适用于连接调试）

2、需要在pom.xml中添加如下依赖插件

<plugin>

<groupId>io.fabric8</groupId>

<artifactId>docker-maven-plugin</artifactId>

<version>0.28.0</version>

<configuration>

<registry>${harbor.address}</registry>

<images>

<image>

<name>${harbor.address}/${project.build.finalName}/${project.build.finalName}:${time.stamp}</name>

<alias>${project.name}</alias>

<build>

<dockerFileDir>${project.basedir}</dockerFileDir>

<tags>

<tag>${time.stamp}</tag>

</tags>

<args>

<JAR\_FILE>target/${project.build.finalName}.jar</JAR\_FILE>

</args>

</build>

</image>

</images>

</configuration>

<dependencies>

<!--该插件需要这个依赖 -->

<dependency>

<groupId>commons-codec</groupId>

<artifactId>commons-codec</artifactId>

<version>1.11</version>

</dependency>

</dependencies>

</plugin>

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>build-helper-maven-plugin</artifactId>

<executions>

<execution>

<id>timestamp-property</id>

<goals>

<goal>timestamp-property</goal>

</goals>

</execution>

</executions>

<configuration>

<name>current.time</name>

<pattern>yyyyMMdd\_HHmmss</pattern>

<timeZone>GMT+8</timeZone>

</configuration>

</plugin>

其中${harbor.address}为harbor服务器地址：

**线上地址为：prod-harbor.yooli-me.com**

**测试环境地址请与测试同学沟通**

3、log日志

目前采用logback-spring.xml日志文件格式，日志会每天自动切割，前一天日志可保存zip/tar.tz/log等文件。Logpath指定为./log。如果有特殊需求或者其他方式请自行与运维同学商量修改脚本。（此日志模板会同步群文件中）

至此docker集成已完成。

# 五 注意事项

1、由于${harbor.address}是需要在pom中注入harbor服务地址。在多环境的时候无法通过在properties中取值放入pom.xml中赋值。目前的做法是通过如下方式

<profiles>

<profile>

<id>dev</id>

<activation>

<activeByDefault>true</activeByDefault>

</activation>

<properties>

<package.environment>src/main/resources/dev</package.environment>

<harbor.address>dev-harbor.com</harbor.address>

</properties>

</profile>

<profile>

<id>test</id>

<properties>

<package.environment>src/main/resources/test</package.environment>

<harbor.address>test-harbor.com</harbor.address>

</properties>

</profile>

<profile>

<id>product</id>

<properties>

<package.environment>src/main/resources/pro</package.environment>

<harbor.address>**prod-harbor.yooli-me.com**</harbor.address>

</properties>

</profile>

</profiles>

如果有更好的方式可以自行解决，并欢迎提出与大家分享

1. 目前部署docker支持测试环境与生产环境

# Jdk兼容docker

Java 8曾经与Docker无法很好地兼容性，现在问题已消失。

注意：我在本文中使用采用GNU GPL v2许可证的OpenJDK官方docker映像。在Oracle Java SE中，这里描述的docker支持功能在更新191中引入。Oracle在2019年4月更改了Java 8更新的许可证，自Java SE 8 Update 211以来商业使用不再免费。

你是否遇到过在docker中运行的基于JVM的应用程序出现“随机”故障？或者也许是一些奇怪的死机？两者都可能是Java 8（仍广泛使用的）中糟糕的docker支持引起的。

Docker使用控制组（cgroups）来限制资源。在容器中运行应用程序时限制内存和CPU绝对是个好主意――它可以阻止应用程序占用整个可用内存及/或CPU，这会导致在同一个系统上运行的其他容器毫无反应。限制资源可提高应用程序的可靠性和稳定性。它还允许为硬件容量作好规划。在Kubernetes或DC/OS之类的编排系统上运行容器时尤为重要。

之前遇到的问题

CPU和内存直接使用宿主机的资源。跳过docker环境

解决方法

方法1：新的Java版本（10及以上版本）已经内置了docker支持功能。但有时升级不是办法，比如说如果应用程序与新JVM不兼容就不行。

方法2：Docker支持还被向后移植到Java 8。不妨检查标记为8u212的最新openjdk映像。我们将内存限制为1G，并使用1个CPU：docker run -ti --cpus 1 -m 1G openjdk：8u212-jdk。

如果由于某种原因不想要看到新的JVM行为，可以使用-XX：-UseContainerSupport来关闭