

什么是 Docker

Docker 使用 Google 公司推出的 Go 语言 进行开发实现，基于 Linux 内核的 cgroup，namespace，以及 AUFS 类的 Union FS 等技术，对进程进行封装隔离，属于 操作系统层面的虚拟化技术。由于隔离的进程独立于宿主和其它的隔离的进程，因此也称其为容器。最初实现是基于 LXC，从 0.7 版本以后开始去除 LXC，转而使用自行开发的 libcontainer，从 1.11 开始，则进一步演进为使用 runC 和 containerd。

Docker 在容器的基础上，进行了进一步的封装，从文件系统、网络互联到进程隔离等等，极大的简化了容器的创建和维护。使得 Docker 技术比虚拟机技术更为轻便、快捷。

下面的图片比较了 Docker 和传统虚拟化方式的不同之处。传统虚拟机技术是虚拟出一套硬件后，在其上运行一个完整操作系统，在该系统上再运行所需应用进程；而容器内的应用进程直接运行于宿主的内核，容器内没有自己的内核，而且也没有进行硬件虚拟。因此容器要比传统虚拟机更为轻便。

传统虚拟化

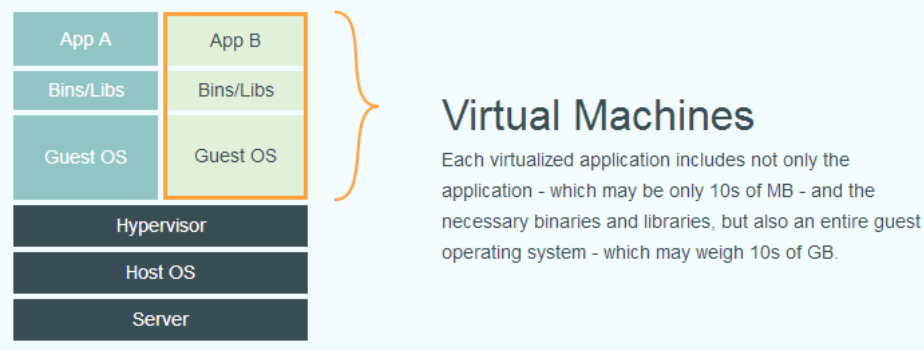


图 1.4.1.1 - 传统虚拟化

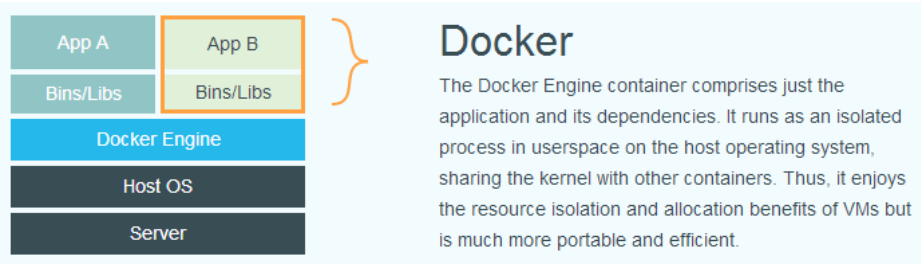


图 1.4.1.2 - Docker

为什么要使用 Docker?

作为一种新兴的虚拟化方式，Docker 跟传统的虚拟化方式相比具有众多的优势。

- 更高效的利用系统资源
- 更快速的启动时间
- 一致的运行环境
- 持续交付和部署
- 更轻松的迁移
- 更轻松的维护和扩展

对比传统虚拟机总结

特性	容器	虚拟机
启动	秒级	分钟级
硬盘使用	一般为 MB	一般为 GB
性能	接近原生	弱于
系统支持量	单机支持上千个容器	一般几十个

了解更多：https://yeasy.gitbooks.io/docker_practice/content/introduction/why.html

引用自：https://yeasy.gitbooks.io/docker_practice/content/introduction/why.html

Spring Boot 与 Docker

Spring Boot简化了Spring应用的开发过程，遵循约定优先配置的原则提供了各类开箱即用（out-of-the-box）的框架配置。另一方面，Spring Boot还具备将代码直接构建为可执行jar包的能力，这个jar包是一个可以独立运行的部署单元。基于以上特性，现在普遍认为Spring Boot提供了一种快速构造微服务(Micro-Service)的能力。

Spring Boot应用通常被构建为一个可单独执行的jar包，将Spring Boot应用容器化为Docker容器镜像并运行，对于自动化部署、运维都是非常有利的。

引用自：<https://www.tianmaying.com/tutorial/spring-boot-docker>

那我们开始吧~

说明

本人小白，linux基础、容器技术尚不扎实 本文仅作为个人爱好的实践成果的记录，且多为引用，如有不恰当之处还请体谅并指正~谢！关于docker的更多知识请移步官方文档和权威指南。

准备

我的环境：

idea 安装docker插件（eclipse可以使用SpringBoot-Docker模板直接生成SpringBoot-Docker项目）

Docker （docker安装：https://yeasy.gitbooks.io/docker_practice/content/install/）

Redis

Maven

MySql

一个简单的Spring Boot 项目, 项目地址:<https://github.com/XxXindyz/springBoot-shiro.git>

集成Redis(session共享)和MySql(数据源)

使用 maven 的 docker-maven-plugin 插件部署

pom.xml 配置

```
<plugin>
  <groupId>com.spotify</groupId>
  <artifactId>docker-maven-plugin</artifactId>
  <version>0.4.13</version>
  <configuration>
    <imageName>springio/${project.artifactId}</imageName>
    <dockerDirectory>src/main/docker</dockerDirectory>
    <resources>
      <resource>
        <targetPath></targetPath>
        <directory>${project.build.directory}</directory>
        <include>${project.build.finalName}.jar</include>
      </resource>
    </resources>
  </configuration>
</plugin>
```

上述pom.xml包含了docker-maven-plugin的配置：

imageName指定了镜像的名字

dockerDirectory指定Dockerfile的位置

resources是指那些需要和Dockerfile放在一起，在构建镜像时使用的文件，一般应用jar包需要纳入

Dockerfile

在pom中dockerDirectory配置的目录创建docker目录，新建Dockerfile文件，关于dockerfile的编写这里不展开，此处贴一下我的

```
FROM frovlvlad/alpine-oraclejdk8:slim VOLUME /tmp ADD springBoot-0.0.1.jar app.jar RUN sh -c 'touch /app.jar' ENV JAVA_OPTS="" ENTRYPOINT
```

这里有个坑：

部署时报错：

```
[ERROR] Failed to execute goal com.spotify:dockerfile-maven-plugin:1.3.6:build (default) on project logbackExample: Could not build image:
java.util.concurrent.ExecutionException: com.spotify.docker.client.shaded.javax.ws.rs.ProcessingException: org.apache.http.client.ClientProtocolException: Cannot retry
request with a non-repeatable request entity: Broken pipe -> [Help 1]
```

这里其实是因为pom.xml中配置 `<imageName>${docker.image.prefix}/${project.artifactId}</imageName>` 使用了pom中的 `<artifactId>springBoot</artifactId>` 然而！docker中使用大写的imageName是会出问题的，因此此处 **千万注意！！** `<artifactId>springBoot</artifactId>` 不能包含大写 如此处springBoot可改为spring-boot

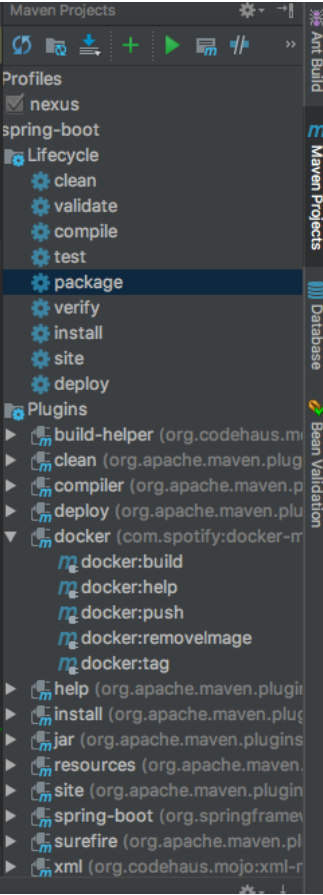
idea集成Docker

参考 <https://www.jetbrains.com/help/idea/docker.html>

eclipse集成Docker

参考 <http://dockone.io/article/437>

本文以idea为例。加入docker-maven-plugin后的MavenProjects:



使用图中docker->docker:build 即可在本地docker中生成一个镜像

当然 运行下列命令也可以在本地Docker中创建一个镜像：

```
$ mvn package docker:build
```

这里我遇到一个还没有解决的坑 导致我无法通过maven插件的方式创建镜像，只能通过docker命令进行创建，该方式后面会讲

问题：使用 `mvn package docker:build` 失败：

```
[ERROR] Failed to execute goal com.spotify:docker-maven-plugin:0.4.13:build (default-cli) on project spring-boot: Exception caught: java.util.concurrent.ExecutionException:
com.spotify.docker.client.shaded.javax.ws.rs.ProcessingException: java.lang.IllegalArgumentException: Host name may not be null -> [Help 1] [ERROR]
```

希望有大佬能帮我解决一下，可能是我环境变量配置的问题。

手动部署

创建镜像

创建一个目录如docker

新建Dockerfile，内容同上

命令行进入该文件夹目录 执行

```
sudo docker build -t xxxindy/spring-boot-app .
```

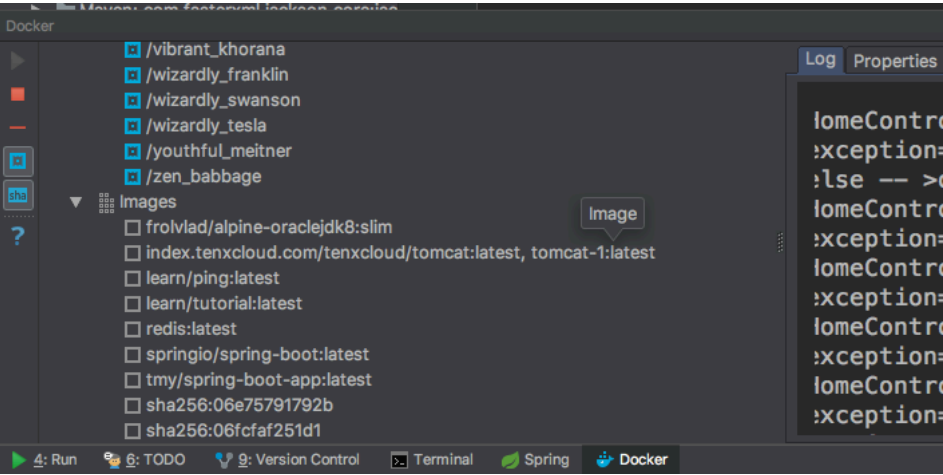
xxxindy/spring-boot-app 为镜像名称

.表示路径 注意这个 . 点！

```
xxxindydeMacBook-Air:docker xxxindy$ sudo docker build -t tmy/spring-boot-app .
Password:
Sending build context to Docker daemon 45.9MB
Step 1/6 : FROM frolvlad/alpine-oraclejdk8:slim
----> 48dafc3ce80c
Step 2/6 : VOLUME /tmp
----> Using cache
----> fc260f277fab
Step 3/6 : ADD springBoot-0.0.1.jar app.jar
----> 6f838f6cfe2c
Step 4/6 : RUN sh -c 'touch /app.jar'
----> Running in b440c8db2336
Removing intermediate container b440c8db2336
----> 42b588308cf2
Step 5/6 : ENV JAVA_OPTS=""
----> Running in 5becca5e5dec
Removing intermediate container 5becca5e5dec
----> 2aea1c6faf62
Step 6/6 : ENTRYPOINT [ "sh", "-c", "java $JAVA_OPTS -Djava.security.egd=file:/dev/./urandom -jar /app.jar" ]
----> Running in 2a842c8c708c
Removing intermediate container 2a842c8c708c
----> 939af95339c4
Successfully built 939af95339c4
Successfully tagged tmy/spring-boot-app:latest
```

镜像创建成功

此时进入idea的docker插件



可以看到新建的镜像

此时其实就可以通过下面的命令创建容器

```
sudo docker run -d -p 8080:8080 --name sample-app xxxindy/spring-boot-app
```

但是由于我的项目在启动时需要redis环境作为session缓存所以这里如果直接启动是会失败的，所以我选择另起一个redis的容器，然后与web容器进行连接。

Redis容器

创建

```
$ sudo docker run --name test-redis -d redis
```

由于本地并没有redis镜像，所以它会自动去dockerhub上拉取

```
Unable to find image 'redis:latest' locally latest: Pulling from library/redis
70e9a6907f10: Pull complete
32f2a4cccab8: Pull complete
...
34446d9c0a72: Pull complete
1f4ff6e27d64: Pull complete
Digest: sha256:68524efa50a33d595d7484de3939a476b38667c7b4789f193761899ca125d016 Status: Downloaded newer image for redis:latest
71c69b5bf62cc4521a43f91869114d598eab5d4826372371909c461e6f024f71
```

查看

```
$ sudo docker ps
```

查看日志，也可以在idea插件中查看

```
$ sudo docker logs
```

会看到redis启动的小蛋糕，兴奋 嘻嘻

配置redis.conf

指定配置

指定配置后，映射本地卷，就可以对数据文件和日志文件的读写位置进行控制

创建redis6379.conf

<https://github.com/XxXindy/GraduationProject.EJUT.2018/blob/master/images/redis6379.conf>

挂载本地卷到容器

```
$ sudo docker run --name myredis -d -v /tmp/x:/data redis redis-server /data/redis6379.conf
```

分析

--name myredis： 给这个容器取名为myredis

-v /tmp/x:/data： 本地将的/tmp/x目录挂载到容器中的/data目录 /tmp/x为本机刚刚创建的自定义redis6379.conf的路径

redis-server /data/redis6379.conf： 使用指定的配置初始化并启动Redis的服务

引用：<http://soft.dog/2016/04/28/redis-docker-config/>

web容器连接redis容器

在连接之前，我们需要知道redis容器的ip

```
$ sudo docker inspect myredis
```

```
{
  "GlobalIPv6Address": "",
  "GlobalIPv6PrefixLen": 0,
  "IPAddress": "172.17.0.3",
  "IPPrefixLen": 16,
  "IPv6Gateway": "",
  "MacAddress": "02:42:ac:11:00:03",
  "Networks": {
    "bridge": {
      "IPAMConfig": null,
      "Links": null,
      "Aliases": null,
      "NetworkID": "3aa76044bf0e9af1c7577c037630b1b8a0c8ec62580a1eb97a22526837bfaf15",
      "EndpointID": "7b483687018885685dcfbbb7f8eb89d419699a55eaa741c752f9fa49d272e7",
      "Gateway": "172.17.0.1",
      "IPAddress": "172.17.0.3",
      "IPPrefixLen": 16,
      "IPv6Gateway": "",
      "GlobalIPv6Address": "",
      "GlobalIPv6PrefixLen": 0,
      "MacAddress": "02:42:ac:11:00:03",
      "DriverOpts": null
    }
  }
}
```

显示myredis的ip为172.17.0.3

修改SpringBoot项目的application.properties

```
spring.redis.host=172.17.0.3
```

很显然，对项目做出修改需要重新打包并更新镜像

```
$ sudo docker build -t tmy/spring-boot-app .
```

用更新后的镜像创建一个连接myredis容器的新容器

```
$ sudo docker run -d -p 8081:8081 --name sample-app --link myredis:redis xxxindy/spring-boot-app:latest
```

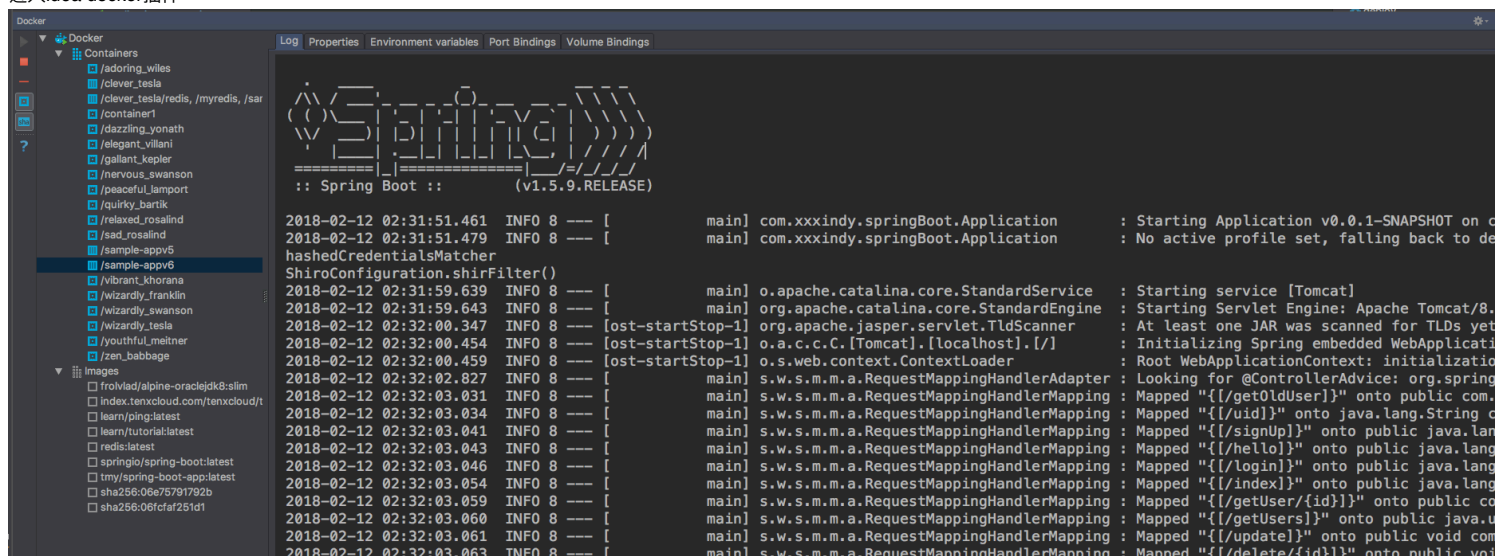
--name sample-app 连接产生的新的容器的名字

--link myredis:redis 表示连接myredis容器

```
$ sudo docker ps
```

将看到名为sample-app的容器

进入idea docker插件



可以看到容器启动，项目正常运行