

Docker服务开放了这个端口，服务器分分钟变肉机！

原创 梦想de星空 macrozheng 2020-08-12 09:02

收录于合集

#mall学习教程（参考篇）

27个

之前有很多朋友提过，当使用 `docker-maven-plugin` 打包SpringBoot应用的Docker镜像时，服务器需要开放 `2375` 端口。由于开放了端口没有做任何安全保护，会引起安全漏洞，被人入侵、挖矿、CPU飙升这些情况都有发生，今天我们来聊聊如何解决这个问题。

问题产生的原因

首先我们要明白问题产生的原因，才能更好地解决问题！

Docker为了实现集群管理，提供了远程管理的端口。Docker Daemon作为守护进程运行在后台，可以执行发送到管理端口上的Docker命令。

当我们修改 `docker.service` 文件，修改启动命令，加入 `-H tcp://0.0.0.0:2375` 时，就会开放 `2375` 端口，且没有任何加密和认证过程，这种方式一般用在内网测试环境。如果你的服务器部署在公网上，任何知道你IP的人，都可以管理这台主机上的容器和镜像，想想就觉得可怕。

解决思路

开放远程管理端口后，没有做任何安全保护导致了这个问题。我们只要使用安全传输层协议（TLS）进行传输并使用CA认证即可。

制作证书及密钥

我们需要使用OpenSSL制作CA机构证书、服务端证书和客户端证书，以下操作均在安装Docker的Linux服务器上进行。

- 首先创建一个目录用于存储生成的证书和密钥；

```
mkdir /mydata/docker-ca && cd /mydata/docker-ca
```

- 创建CA证书私钥，期间需要输入两次用户名和密码，生成文件为 **ca-key.pem** ；

```
openssl genrsa -aes256 -out ca-key.pem 4096
```

- 根据私钥创建CA证书，期间需要输入上一步设置的私钥密码，生成文件为 **ca.pem** ；

```
openssl req -new -x509 -days 365 -key ca-key.pem -sha256 -subj "/CN=*" -out ca.pem
```

- 创建服务端私钥，生成文件为 **server-key.pem** ；

```
openssl genrsa -out server-key.pem 4096
```

- 创建服务端证书签名请求文件，用于CA证书给服务端证书签名，生成文件 **server.csr** ；

```
openssl req -subj "/CN=*" -sha256 -new -key server-key.pem -out server.csr
```

- 创建CA证书签名好的服务端证书，期间需要输入CA证书私钥密码，生成文件为 **server-cert.pem** ；

```
openssl x509 -req -days 365 -sha256 -in server.csr -CA ca.pem -CAkey ca-key.pem -CAcreateserial -out server-cert.pem
```

- 创建客户端私钥，生成文件为 **key.pem** ；

```
openssl genrsa -out key.pem 4096
```

- 创建客户端证书签名请求文件，用于CA证书给客户证书签名，生成文件 **client.csr** ；

```
openssl req -subj "/CN=client" -new -key key.pem -out client.csr
```

- 为了让秘钥适合客户端认证，创建一个扩展配置文件 **extfile-client.cnf** ；

```
echo extendedKeyUsage = clientAuth > extfile-client.cnf
```

- 创建CA证书签名好的客户端证书，期间需要输入CA证书私钥密码，生成文件为 **cert.pem** ；

```
openssl x509 -req -days 365 -sha256 -in client.csr -CA ca.pem -CAkey ca-key.pem -CAcreateserial -o
```

- 删除创建过程中多余的文件；

```
rm -rf ca.srl server.csr client.csr extfile-client.cnf
```

- 最终生成文件如下，有了它们我们就可以进行基于TLS的安全访问了。

```
ca.pem CA证书
ca-key.pem CA证书私钥
server-cert.pem 服务端证书
server-key.pem 服务端证书私钥
cert.pem 客户端证书
key.pem 客户端证书私钥
```

配置Docker支持TLS

- 用vim编辑器修改docker.service文件；

```
vi /usr/lib/systemd/system/docker.service
```

- 修改以 **ExecStart** 开头的配置，开启TLS认证，并配置好CA证书、服务端证书和服务端私钥，修改内容如下；

```
ExecStart=/usr/bin/dockerd -H fd:// -H tcp://0.0.0.0:2375 --tlsverify --tlscacert=/mydata/docker-ca
```

- 重启Docker服务，这样我们的Docker服务就支持使用TLS进行远程访问了！

```
systemctl daemon-reload && systemctl restart docker
```

客户端访问

接下来我们将使用 **docker-maven-plugin** 来打包Docker镜像，使用的代码为原来的 **mall-tiny-docker** 例子。

- 直接使用 **docker-maven-plugin** 打包试试，由于我们的插件版本有点低，使用新一点版本的Docker会出现如下问题，升级到 **1.2.2** 版本解决该问题；

```
[ERROR] Failed to execute goal com.spotify:docker-maven-plugin:1.1.0:build (build-image) on project
[ERROR] at [Source: UNKNOWN; line: -1, column: -1] (through reference chain: java.util.LinkedHash
[ERROR] -> [Help 1]
```

- 修改完版本后打包，发现TLS不再支持 `http` 了，需要改用 `https`，修改 `<dockerHost>` 配置为 `https`；

```
[ERROR] Failed to execute goal com.spotify:docker-maven-plugin:1.2.2:build (build-image) on project
```

- 修改完成后再次打包，继续失败，需要添加对应的客户端证书才能访问；

```
[ERROR] Failed to execute goal com.spotify:docker-maven-plugin:1.2.2:build (build-image) on project
```

- 将如下文件复制到指定目录，这里复制到了 `I:\developer\env\docker-ca`；

```
ca.pem CA证书
cert.pem 客户端证书
key.pem 客户端证书私钥
```

- 然后将该目录配置在插件的 `<dockerCertPath>` 节点下，最终插件配置如下；

```
<plugin>
  <groupId>com.spotify</groupId>
  <artifactId>docker-maven-plugin</artifactId>
  <version>1.2.2</version>
  <executions>
    <execution>
      <id>build-image</id>
      <phase>package</phase>
      <goals>
        <goal>build</goal>
      </goals>
    </execution>
  </executions>
  <configuration>
    <imageName>mall-tiny/${project.artifactId}:${project.version}</imageName>
    <dockerHost>https://192.168.3.101:2375</dockerHost>
    <baseImage>java:8</baseImage>
```

```

<entryPoint>["java", "-jar", "/${project.build.finalName}.jar"]
</entryPoint>

<dockerCertPath>I:\developer\env\docker-ca</dockerCertPath>

<resources>
    <resource>
        <targetPath></targetPath>
        <directory>${project.build.directory}</directory>
        <include>${project.build.finalName}.jar</include>
    </resource>
</resources>
</configuration>
</plugin>

```

- 再次打包镜像，发现已经可以成功打包镜像，从此我们的 2375 端口终于可以安全使用了！

```
[INFO] Building image mall-tiny/mall-tiny-docker:0.0.1-SNAPSHOT
```

```
Step 1/3 : FROM java:8
```

```
---> d23bdf5b1b1b
```

```
Step 2/3 : ADD /mall-tiny-docker-0.0.1-SNAPSHOT.jar //
```

```
---> 5cb5a64ccedd
```

```
Step 3/3 : ENTRYPOINT ["java", "-jar", "/mall-tiny-docker-0.0.1-SNAPSHOT.jar"]
```

```
---> Running in 5f3ceefdd974
```

```
Removing intermediate container 5f3ceefdd974
```

```
---> ee9d0e2b0114
```

```
ProgressMessage{id=null, status=null, stream=null, error=null, progress=null, progressDetail=null}
```

```
Successfully built ee9d0e2b0114
```

```
Successfully tagged mall-tiny/mall-tiny-docker:0.0.1-SNAPSHOT
```

```
[INFO] Built mall-tiny/mall-tiny-docker:0.0.1-SNAPSHOT
```

```
[INFO] -----
```

```
[INFO] BUILD SUCCESS
```

```
[INFO] -----
```

```
[INFO] Total time: 20.550 s
```

```
[INFO] Finished at: 2020-07-31T15:02:15+08:00
```

```
[INFO] Final Memory: 50M/490M
```

```
[INFO] -----
```

官方文档：<https://docs.docker.com/engine/security/https/>

项目源码地址

<https://github.com/macrozheng/mall-learning/tree/master/mall-tiny-docker>

推荐阅读

- [Mall 电商实战项目发布重大更新，全面支持SpringBoot 2.3.0！](#)
- [秒杀商品超卖事故：Redis分布式锁请慎用！](#)
- [被我用烂的DEBUG调试技巧，专治各种搜索不到的问题！](#)
- [我扒了半天源码，终于找到了Oauth2自定义处理结果的最佳方案！](#)
- [10天竟然只写了一行代码，谁的锅？](#)
- [别再if-else走天下了，整个注解多优雅！](#)
- [Elasticsearch 升级 7.x 版本后，我感觉掉坑里了！](#)
- [肝了两天IntelliJ IDEA 2020，解锁11种新姿势，真香！！！！](#)
- [一个不容错过的Spring Cloud实战项目！](#)
- [我的Github开源项目，从0到20000 Star！](#)



欢迎关注，点个在看

收录于合集 #mall学习教程（参考篇） 27

上一篇

给Swagger换了个新皮肤，瞬间高大上了！

下一篇

居然有人想白嫖我的日志，赶紧开启安全保护压压惊！

阅读原文

喜欢此内容的人还喜欢

项目中到底该不该用Lombok?
macrozheng

