使用Docker Secrets存储.NET Core应用密码

Published on Apr 11, 2021 UTC by Wang Ye <https://wangye.org>

大多数情况下我们开发的应用可能需要与其他服务进行交互，比如数据库服务、第三方服务等等，出于安全考虑，这些服务的交互都需要发起端提供认证凭据，这里可能是账号密码、API Key等，这里就涉及到认证凭据的保管问题，很长一段时间我们都是将认证凭据写死到配置文件中，这样做一开始是简单可行，但是随后你就会发现带来了一系列的问题：首先你的所有开发环境和生产环境都需要配置成一样的，比如生产环境数据库账号和密码和开发环境数据库账号和密码最好是一致的，否则部署到生产环境则需要手动修改这个认证凭据，当然你也可以通过配置分离或者环境变量控制来避免这个问题；其次是代码仓库提交，尤其是开源项目，至今GitHub公共库仍然存在大量配置文件明文保管着敏感凭据；最后分布式开发或者交付麻烦，需要团队合作或者直接提供给终端用户的项目，团队成员或者终端用户需要修改为本地环境凭据。

1 .NET Core与secrets.json

.NET Core以其易开发、高性能及跨平台著称，刚开始我也是将凭据保存在配置文件`appsettings.json`中，对于生产环境通过环境变量`ASPNETCORE_ENVIRONMENT`来控制，比如设置这个环境变量值为`Production`，那么应用将会采用`appsettings.Production.json`覆盖原有的配置，类似于下面这样结构，下面的配置文件如果和上面配置文件存在相同的值，那么下面的值将会覆盖上面配置文件的值，.NET Core应用的加载顺序则是从上往下依次加载。

```
|--appsettings.json
  |--appsettings.Production.json
```

这样我们可以将开发环境认证凭据保存在appsettings.Development.json里面，生产环境与开发环境切换丝滑，这个适合一个人的私有项目，如果多人合作开发或者提交到公共代码库，那么我们在文章开头提到的问题依然存在。

1.1 启用secrets机密存储支持

针对这个痛点，微软提供了保存机密(secrets)信息的办法，cmd切换到项目路径下，使用下面的命令建立secrets存储支持（适用于.NET Core 3.0.100及以后版本）：

```
dotnet user-secrets init
```

执行完命令以后，项目文件.csproj会多一行配置信息如下：

```
<PropertyGroup>
  <UserSecretsId>79a3edd0-2092-40a2-a04d-dcb46d5ca9ed</UserSecretsId>
</PropertyGroup>
```

这里的UserSecretsId内容是随机生成的GUID标识符，实际可能与示例不一样。

1.2 保存机密信息

比如我们应用有个第三方服务Movies，其中认证凭据是API Key，原先在appsettings.json中如下配置：

```
{
  "Movies": {
    "ServiceApiKey": "12345"
  }
}
```

很明显12345是我们需要保护的机密信息，通过以下命令即可完成保存：

```
dotnet user-secrets set "Movies:ServiceApiKey" "12345"
```

可以发现，这条命令支持层次配置，比如在json配置文件中ServiceApiKey是Movies的子节，那么我们就可以以冒号:来分隔父节或者子节。

执行完命令后就可以删除原先存在于appsettings.json的机密信息，可能有读者会问，我们应用需要修改吗？appsettings.json里面的机密信息的删除后还需要额外的代码支持去secrets里面获取机密信息吗？这里统统不需要，也就是说这样迁移配置就已经完成了，对于原理我简单说明一下：实际上这项技术的实现是在刚才配置节中插入了一个名叫secrets.json的机密信息配置文件，配置加载顺序变成以下：

```
|--appsettings.json
  |--appsettings.Production.json
  |--secrets.json
```

同样.NET Core依次从上往下加载，下面的配置会覆盖上面的配置，所以又如丝般顺滑，等等，这个和一开始根据环境来划分appsettings有区别吗？当然有区别，最显著的区别是secrets.json不保存在项目路径下，也就是说你可以放心提交代码而不必担心认证凭据泄露，对于Windows操作系统来说一般位于路径：

```
%APPDATA%\Microsoft\UserSecrets\<user_secrets_id>\secrets.json
```

这里<user_secrets_id>就是一开始启用支持时生成的UserSecretsId值，对于Linux等类Unix系统一般位于路径：

```
~/.microsoft/usersecrets/secrets.json
```

有了这项配置，那么之前遇到的问题也就能顺利解决啦，更多内容可以直接参考微软官方文档。

2 使用Docker Secrets

2.1 传统方式

实际上对于Docker容器.NET Core官方已经支持的很好，基于前面关于`secrets.json`存储位置的认识，我们的`docker-compose.yaml`内容参考微软官方文档可以像这样：

```
version: "3.4"

services:
  web:
    environment:
      - ASPNETCORE_ENVIRONMENT=Development
      - ASPNETCORE_URLS=https://+:443;http://+:80
    ports:
      - "80"
      - "443"
    volumes:
      -
${APPDATA}/Microsoft/UserSecrets:/root/.microsoft/usersecrets:ro
      - ${APPDATA}/ASP.NET/Https:/root/.aspnet/https:ro
```

这样就将宿主Windows下的用户凭据路径映射到Docker容器的`/root/.microsoft/usersecrets`路径中，对于类Unix的映射挂载路径可以类推，然后把凭据文件`secrets.json`上传到这个路径下即可。

2.2 使用Docker Secrets

Docker Secrets提供一种保管认证凭据的方式，进一步减少密码暴露，保障系统安全可靠。通过采用Docker Secrets技术，容器在启用时Docker将自动挂载一个名为`/run/secrets`路径，其中认证凭据将以一个个"文件"保存在这里，虽然说是文件，实际上是映射到内存的，应用读取速度不受影响。 比如说我们有个`Movies`的API Key：`12345`，那么在`docker-compose.yaml`同路径下可以通过下面命令创建一个包含Key的文件：

```
echo -n 12345 > .secrets.txt
```

刚才的配置文件可以改成如下：

```
version: "3.4"

services:
  web:
    environment:
      - ASPNETCORE_ENVIRONMENT=Development
      - ASPNETCORE_URLS=https://+:443;http://+:80
    ports:
      - "80"
      - "443"
    volumes:
      -
${APPDATA}/Microsoft/UserSecrets:/root/.microsoft/usersecrets:ro
      - ${APPDATA}/ASP.NET/Https:/root/.aspnet/https:ro
    secrets:
      - ServiceApiKey

secrets:
  ServiceApiKey:
    file: ./.secrets.txt
```

重新编译并在后台拉起docker：

```
docker-compose build
docker-compose up -d
```

进入到容器内部，你就会发现`/run/secrets`路径，以及路径下的ServiceApiKey文件，文件内容正式我们需要保护的key。

当然对于.NET Core应用来说这远远是不够的，我们必须让应用能够感知到Docker Secrets，使用nuget安装包`Microsoft.Extension.Configuration.KeyPerFile`，并且增加配置如下

（ASP.NET应用位于Program.cs文件）：

```
.ConfigureAppConfiguration((hostingContext, config) =>
{
    config.AddKeyPerFile(directoryPath: "/run/secrets", optional:
true);
})
```

这样就完成了配置，但是总感觉有些不对劲，对于层次关系的处理方式这里似乎没有解决，是的，我搜索网络找到了Josef Ottosson的解决方案[《Dotnet Core ConfigurationProvider for Docker Swarm Secrets》](), 他通过重新实现IConfigurationBuilder，并将冒号(:)转换为下划线较完美的解决了问题，这样通过Josef Ottosson的解决方案，再来看1.2所讲的例子：Movies:ServiceApiKey，可以进行如下配置：

```
version: "3.4"

services:
  web:
    # other settings
    secrets:
      - Movies_ServiceApiKey

secrets:
  Movies_ServiceApiKey:
    file: ./.secrets.txt
```

好了，就先介绍这么多，感谢阅读。

External References (Links)

微软官方文档
https://docs.microsoft.com/en-us/aspnet/core/security/app-secrets
微软官方文档
https://docs.microsoft.com/en-us/dotnet/architecture/containerized-

lifecycle/design-develop-containerized-apps/build-aspnet-core-applications-linux-containers-aks-kubernetes
《Dotnet Core ConfigurationProvider for Docker Swarm Secrets》
https://josef.codes/dotnet-core-configurationprovider-for-docker-swarm-secrets/

