How to restore nuget packages from an Azure DevOps Private Feed when building a Docker image

2020-07-14 — 5 min read

#csharp #docker #nuget #containers #windows

Let's go for a quick post this time.

Imagine you're trying to create a Docker image from an application and it is using some custom nugets. Those custom nugets are hosted in your private Azure DevOps Feed. And when you try to build the image you get one of the following errors:

```
/src/MyConsoleApplication.csproj : error NU1101: Unable to find package MyOwr
  Failed to restore /src/MyConsoleApplication.csproj (in 2.37 sec).
```

```
C:\Program Files\dotnet\sdk\3.1.301\NuGet.targets(128,5): error : Unable to l
```

```
C:\Program Files\dotnet\sdk\3.1.301\NuGet.targets(128,5): error :   Response
```

That's a pretty common pain point when trying to create a Docker image that uses a private Azure DevOps Feed.

To solve this problem there are two possible solutions:

- Place a NuGet.Config file inside your application and use it when creating

the image.
- Use the Azure Artifacts Credentials Provider

The first option is the older one and used to be the way to do it.
The second one is the most recent one and in my opinion a better way to do it.

I'm going to show you how you can fix it using both Linux and Windows Containers. But first I'm going to set everything up on the Azure Devops end.

I did the following steps on my Azure DevOps account:

1- Create a private Azure DevOps Feed named ***"my-very-private-feed"***

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <packageSources>
    <clear />
    <add key="my-very-private-feed" value="https://pkgs.dev.azure.com/cpn/_pa
  </packageSources>
</configuration>
```

2- Push a custom Nuget named ***"MyOwn.EmailService"*** into my the Azure DevOps feed.

3- Create a .NETCore 3.1 console app that uses the ***"MyOwn.EmailService"*** nuget

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>netcoreapp3.1</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="MyOwn.EmailService" Version="1.0.0" />
  </ItemGroup>

</Project>
```

4 - Create a DockerFile for the app

```
FROM mcr.microsoft.com/dotnet/core/runtime:3.1-buster-slim AS base
WORKDIR /app

FROM mcr.microsoft.com/dotnet/core/sdk:3.1-buster AS build

WORKDIR /src
COPY ["MyConsoleApplication.csproj", ""]
RUN dotnet restore "./MyConsoleApplication.csproj"
COPY . .
WORKDIR "/src/."
RUN dotnet build "MyConsoleApplication.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "MyConsoleApplication.csproj" -c Release -o /app/publish

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "MyConsoleApplication.dll"]
```

And when everything is set it up if I try to built the image it fails miserably with a `401` error…
Let's see how to fix it.

Scenario 1 : Using a NuGet.Config

Create a NuGet.Config and place it within the app.
You also need to create a PAT *(Personal Access Token)* to authenticate against the private feed.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
```

```
  <packageSources>
    <add key="nuget.org" value="https://api.nuget.org/v3/index.json" protocol
    <add key="my-very-private-feed" value="https://pkgs.dev.azure.com/cpn/_pa
  </packageSources>
  <packageSourceCredentials>
    <my-very-private-feed>
      <add key="Username" value="my-user@outlook.com" />
      <add key="ClearTextPassword" value="put-your-pat-here" />
    </my-very-private-feed>
  </packageSourceCredentials>
</configuration>
```

Change the Dockerfile to use the NuGet.Config file when trying to restore the nugets:

```
FROM mcr.microsoft.com/dotnet/core/runtime:3.1-buster-slim AS base
WORKDIR /app

FROM mcr.microsoft.com/dotnet/core/sdk:3.1-buster AS build

WORKDIR /src
COPY ["MyConsoleApplication.csproj", ""]
COPY NuGet.Config ./

RUN dotnet restore --configfile NuGet.Config "./MyConsoleApplication.csproj"
COPY . .
WORKDIR "/src/."
RUN dotnet build "MyConsoleApplication.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "MyConsoleApplication.csproj" -c Release -o /app/publish

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "MyConsoleApplication.dll"]
```

And it everything works flawlessly.

Scenario 2: Using the Artifacts Credential Provider

The Azure Artifacts Credentials provider is an executable that eases the acquisition of credentials needed to restore NuGet packages.
You can read more about it here

In that scenario we are going to use a bash script to install the credential provider, and also we are going to specify some environment variables for the provider to work.

We need the following variables:

- NUGET_CREDENTIALPROVIDER_SESSIONTOKENCACHE_ENABLED: Controls whether or not the session token is saved to disk. If false, the Credential Provider will prompt for auth every time.
- VSS_NUGET_EXTERNAL_FEED_ENDPOINTS: A JSON that contains an array of service endpoints, usernames and access tokens. The provider authenticates against all the providers found inside the array, so if you want you can have multiple private feeds.

Also In the *"dotnet restore"* step we need to specify our private feed. If we don't specify it the credential provider will try to restore the nugets using only the default feed.

The Dockerfile looks like this:

```
FROM mcr.microsoft.com/dotnet/core/runtime:3.1-buster-slim AS base
WORKDIR /app

FROM mcr.microsoft.com/dotnet/core/sdk:3.1-buster AS build

ENV DOTNET_SYSTEM_NET_HTTP_USESOCKETSHTTPHANDLER=0
ENV NUGET_CREDENTIALPROVIDER_SESSIONTOKENCACHE_ENABLED true
ENV VSS_NUGET_EXTERNAL_FEED_ENDPOINTS {\"endpointCredentials\": [{\"endpoint\

-feed/nuget/v3/index.json\", \"username\":\"my-user@outlook.com\", \"passwor
```

```
RUN wget -qO- https://raw.githubusercontent.com/Microsoft/artifacts-credprovi

WORKDIR /src
COPY ["MyConsoleApplication.csproj", ""]

RUN dotnet restore "./MyConsoleApplication.csproj" -s "https://pkgs.dev.azure

 -s "https://api.nuget.org/v3/index.json"
COPY . .
WORKDIR "/src/."
RUN dotnet build "MyConsoleApplication.csproj"  -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "MyConsoleApplication.csproj" -c Release -o /app/publish

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "MyConsoleApplication.dll"]
```

Windows Containers

And what about the “long forgotten” **Windows Containers?**
Let’s try to replicate the same scenarios but these time using Windows Containers.

Scenario 1 : Using the Nuget.Config with a nanoserver image

The only difference with the Linux scenario is that I’m using a windows nanoserver image

```
FROM mcr.microsoft.com/dotnet/core/runtime:3.1-nanoserver-1903 AS base
WORKDIR /app

FROM mcr.microsoft.com/dotnet/core/sdk:3.1-nanoserver-1903 AS build
```

```
WORKDIR /src
COPY ["MyConsoleApplication.csproj", ""]
COPY NuGet.Config ./

RUN dotnet restore --configfile NuGet.Config "./MyConsoleApplication.csproj"
COPY . .
WORKDIR "/src/."
RUN dotnet build "MyConsoleApplication.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "MyConsoleApplication.csproj" -c Release -o /app/publish

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "MyConsoleApplication.dll"]
```

It works without any hitch.

Scenario 2: Using the Artifacts Credential Provider

That scenario is the real pain!

We are running on windows containers, so we cannot use the bash script to install the provider, instead of using the bash script we are going to use another script written in Powershell.
That's not a problem at all because the nanoserver image comes with Powershell Core already installed.

The real problem is that the credential provider installer doesn't play nice with the nanoserver image. The solution is to install the credentials provider in a **windows server core** image and copy the folder where the credential provider is installed into the nanoserver image.

```
# escape=`

# Install the cred provider in a separate stage based on Windows Server Core
# due to https://github.com/microsoft/artifacts-credprovider/issues/201
```

```
FROM mcr.microsoft.com/powershell as credproviderinstaller
SHELL ["pwsh", "-Command", "$ErrorActionPreference = 'Stop'; $ProgressPrefere
RUN Invoke-WebRequest https://raw.githubusercontent.com/microsoft/artifacts-c
OutFile installcredprovider.ps1; `
    .\installcredprovider.ps1; `
    del installcredprovider.ps1


FROM mcr.microsoft.com/dotnet/core/sdk:3.1-nanoserver-1903 AS build
ENV DOTNET_SYSTEM_NET_HTTP_USESOCKETSHTTPHANDLER=0
ENV NUGET_CREDENTIALPROVIDER_SESSIONTOKENCACHE_ENABLED true
ENV VSS_NUGET_EXTERNAL_FEED_ENDPOINTS `
    "{`"endpointCredentials`": [{`"endpoint`":`"https://pkgs.dev.azure.com/cp

feed/nuget/v3/index.json`", `"username`":`"myUser@outlook.com`", `"password`'


# Copy cred provider from installer stage
COPY --from=credproviderinstaller C:\Users\ContainerAdministrator\.nuget\plug

WORKDIR /src
COPY ["MyConsoleApplication.csproj", ""]
RUN dotnet restore "./MyConsoleApplication.csproj" -s "https://pkgs.dev.azure

-s "https://api.nuget.org/v3/index.json"
COPY . .
WORKDIR "/src/."
RUN dotnet build "MyConsoleApplication.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "MyConsoleApplication.csproj" -c Release -o /app/publish

FROM mcr.microsoft.com/dotnet/core/runtime:3.1-nanoserver-1903 AS base
WORKDIR /app

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "MyConsoleApplication.dll"]
```

At the end of the day, it doesn’t matter if you’re working with windows or linux

containers you can use both solutions indiscriminately.
Using the credentials provider solution with Windows container is more painful because of these weird bug, but I'm pretty confident that Microsoft is going to fix it in a near future.
I used during many years the NuGet.Config solution but nowadays I prefer to use the Credential Provider because it one less file to manage in my repository.

READ OTHER POSTS

← Trying to automate Az... Why and how you sho... →