Agenda: Azure Artifacts

- What are Artifacts
- Public and Download Build Artifacts
- Publish and Download Pipeline Artifacts
- Working with Feed and NuGet Packages
- Upstream Sources
- Public NuGet Package from Pipeline to NuGet Feeds
- Views
- Share Packages Publicly

What are Artifacts?

An artifact is a deployable component of your application. These components can then be deployed to one or more environments.

How do we get an artifact? There are different ways to create and retrieve artifacts, and not every method is appropriate for every situation.

1. Build
2. Azure Repos
3. GitHub
4. TFVC
5. Azure Artifacts
6. Azure Containers
7. Docker Hub
8. Jenkins

The most common and most used way to get an artifact within the release pipeline is to use a **build artifact**. The build pipeline compiles, tests, and eventually produces an **immutable package**, which is stored in a secure place (storage account, database etc.).

The release pipeline then uses a secure connection to this **secured place to get the build artifact** and perform additional actions to deploy this to an environment.

What is a Package?

A package is a formalized way of creating a **distributable unit of software** artifacts that can be consumed from another software solution. The package describes the content it contains and usually provides **additional metadata**. This additional information uniquely identifies the individual packages and to be **self-descriptive**. It helps to better store packages in **centralized locations** and

consume the contents of the package in a predictable manner. In addition, it **enables tooling** to manage the packages in the software solution.

Packaging Formats

- **NuGet:** A NuGet package is essentially a compressed folder structure with published **.NET** project files in ZIP format and has the **.nupkg** extension.
- **NPM**: A NPM package is a file or folder that contains **JavaScript** files and a **package.json** file describing the metadata of the package.
- **Maven**: Each **Java** based projects package has It has a Project Object Model file describing the metadata of the project
- **PyPI**: The Python Package Index, abbreviated as PyPI and also known as the Cheese Shop, is the official third-party software repository for Python.
- **Docker:** Docker packages are called images and contain complete and self-contained deployments of components.

Package feeds

- Packages should be stored in a centralized place for distribution and consumption by others to take dependencies on the components it contains. The centralized storage for packages is most commonly called a package feed.
- Each package type has its own type of feed.
- Package feeds offer versioned storage of packages. A certain package can exist in multiple versions in the feed, catering for consumption of a specific version.
- There are NuGet feeds, NPM feeds, Maven repositories, PyPi feed and Docker registries.
- The feeds can be exposed in public or private to distinguish in visibility. Public feeds can be consumed by anyone.

Publid Feeds:

Package type	Package source	URL
NuGet	NuGet Gallery	**https://nuget.org**
NPM	NPMjs	**https://npmjs.org**
Maven	Maven	**https://search.maven.org**
Docker	Docker Hub	**https://hub.docker.com**
Python	Python Package Index	**https://pypi.org**

The table above does not contain an extensive list of all public sources available. There are other public package sources for each of the types.

Private Feeds:

There are two options for private feeds:

1. Self hosted
2. SaaS Service

The following table contains a non-exhaustive list of self-hosting options and SaaS offerings to privately host package feeds for each of the types covered

Package type	Self-hosted private feed	SaaS private feed
NuGet	NuGet server	Azure Artifacts, MyGet
NPM	Sinopia, cnpmjs.org, Verdaccio	NPMjs.org, MyGet, Azure Artifacts
Maven	Nexus, Artifactory, Archivia	Azure Artifacts, Bintray, JitPack
Docker	Portus, Quay, Harbor	Docker Hub, Azure Container Registry, Amazon Elastic Container Registry
Python	PyPI Server	Gemfury

Working with Feed and NuGet Packages

A feed is a container for packages. You **consume and publish** packages through a particular feed.

Artifacts → Create Feed

In dialog box

- Give the feed a name.
- Choose who can read and contribute (or update) packages in your feed.
- Choose the upstream sources for your feed.
- When you're done, select **Create**.

Step 1: Create a Feed

A feed is a container for packages. You consume and publish packages through a particular feed.

1. Azure Artifacts → **+ Create Feed** → Name="DotNetPackages" → Create

Step2: Create a NuGet Package

2. Create a **.NET Standard Classlibrary** Project.
3. Edit the Project file (*.csproj) as below

```
<Project Sdk="Microsoft.NET.Sdk">
 <PropertyGroup>
  <TargetFramework>netstandard2.0</TargetFramework>
```

```
  <PackageId>HelloWorldApp_Library</PackageId>
  <Version>1.0.0</Version>
  <Authors>Sandeep</Authors>
  <Company>Deccansoft</Company>
  <GeneratePackageOnBuild>true</GeneratePackageOnBuild>
 </PropertyGroup>
</Project>
```

4. Build the project to generate .\bin\Debug**HelloWorldApp_Library.1.0.0.nupkg** file.

Step 3: Publish a Package from the project folder to Feed.

5. Download and install the Credential Provider from https://github.com/microsoft/artifacts-credprovider#azure-artifacts-credential-provider → Click on PowerShell helper script → Convert to RAW format and Save
   a. Save the downloaded PowerShell Script as d:\installcredprovider.ps1
   b. Open Command Promt in Administrator Mode
   c. D:\> Powershell .\installcredprovider.ps1

6. Add a new file to project: **nuget.config**

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
 <packageSources>
  <clear />
  <add key="DotNetPackages" value="https://pkgs.dev.azure.com/DotNet Packages/_packaging/DotNet Packages/nuget/v3/index.json" />
 </packageSources>
</configuration>
```

7. Execute the following command from the same folder where we have saved **nuget.config**

   dotnet nuget push --interactive --source "DotNetPackages" --api-key az

   D:\Temp\ClassLibrary1\bin\Debug\HelloWorldApp_Library.1.0.0.nupkg

   Provide the required credentials

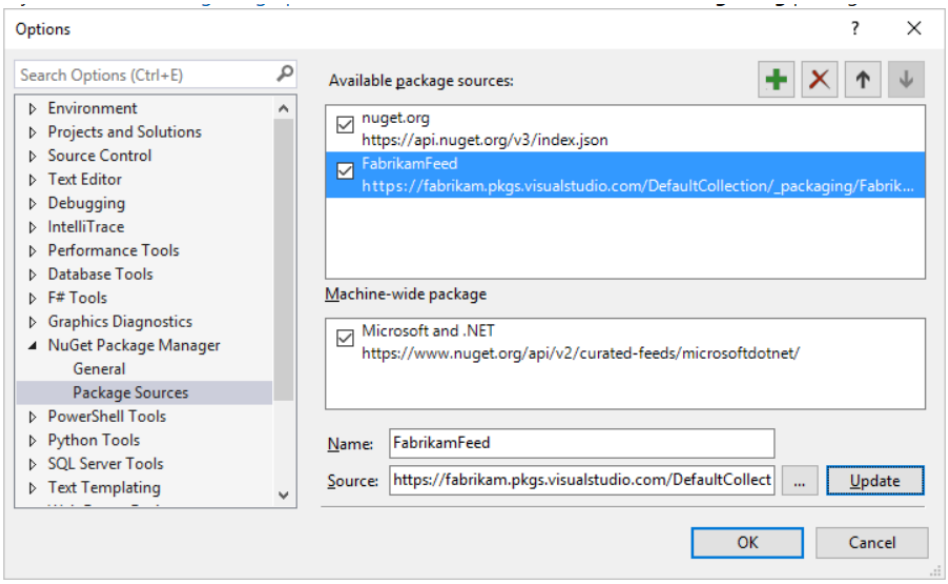
Result: Package is now successfully installed in the Feed.

Step 4: Consume your package in Visual Studio

To consume NuGet packages from a feed, add the feed's NuGet endpoint as a package source in Visual Studio.

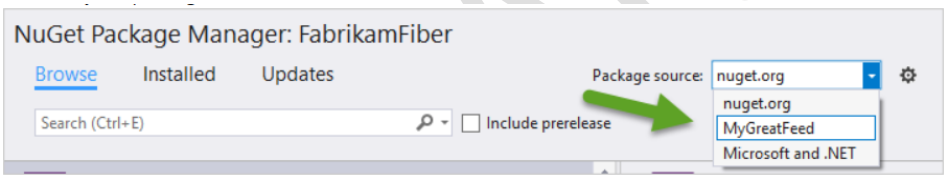
8. Go to your feed → Select **Connect to feed**: → **Visual Studio** tab

9. Visual Studio → Tools → Options → Expand NuGet Package Manager → Package Sources. Select the green plus in the upper-right corner and enter the name and source URL → OK



Step 5: Consume the package

1. Any Visual Studio Project → Right-click **References** →Select **Manage NuGet Packages** → In the **Package source** drop-down list, select your feed.
2. Look for your package in the list.



Secure and share packages using feed permissions

By default, the Project Collection Build Service is a Contributor and your project team is a Reader.

Permission	Reader	Collaborator	Contributor	Owner
List and restore/install packages	✓	✓	✓	✓
Save packages from upstream sources		✓	✓	✓
Push packages			✓	✓
Unlist/deprecate packages			✓	✓

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

Delete/unpublish package				✓
Edit feed permissions				✓
Rename and delete feed				✓

Editing permissions for a feed

With your feed selected, select **Edit feed** (the gear icon) → Select **Permissions** tab

In the edit feed dialog:

- Choose to make each person or team an Owner, Contributor, Collaborator, or Reader.
- When you're done, select **Save**.

Upstream Sources

Upstream sources:

- When combining private and public sources, the order of resolution of the sources becomes important.
- One way to specify multiple packages sources is by choosing a primary source and specifying an upstream source. The package manager will evaluate the primary source first and switch to the upstream source when the package is not found there.
- The upstream source might be one of the official public sources or a private source. The upstream source could refer to another upstream source itself, creating a chain of sources.
- A typical scenario is to use a private package source referring to an public upstream source for one of the official feeds. This effectively enhances the packages in the upstream source with packages from the private feed, avoiding the need to publish private packages in a public feed.
- A source that has an upstream source defined may download and **cache the packages** that were requested it does not contain itself.

- Upstream sources enable you to use a **single feed** to store both the packages you **produce** and the packages you **consume** from "remote feeds": both public feeds (e.g. npmjs.com, nuget.org, Maven Central, and PyPI) and authenticated feeds (i.e. other Azure DevOps Services feeds in your organization or in organizations in your Azure Active Directory (AAD) tenant).
- Once you've enabled an upstream source, any user connected to your feed can install a package from the remote feed, **and your feed will save a copy**.

Use a single feed on the client

In order for your feed to provide deterministic restore, it's important to ensure that your package feed configuration file—your .npmrc or nuget.config—references only your Azure Artifacts feed with upstream sources enabled. For NuGet, the <packageSources> section should look like:

Step1: Steps to Create a feed with upstream sources enabled

1. Azure Artifacts → + Create feed → Name="DeccansoftFeed"… **Check Include packages from common public sources** → Create

Step2: Replace the public registry in configuration files with the Azure Artifacts feed

1. Azure Artifacts → select Feed → Select NuGet.exe Tab → Copy the XML snippet under Project Setup

2. In VS, create a new file named **nuget.config** in the root of your project → Paste the XML snippet.

```
<packageSources>
 <clear />
 <add key="DeccansoftFeed"
value="https://pkgs.dev.azure.com/demoorg/_packaging/DeccansoftFeed/nuget/v3/index.json" />
</packageSources>
```

3. Clear your local package cache:

```
nuget locals -clear all
or
Visual Studio → Tools → NuGet Package Manager → General → Click Clear All NuGet Cache(s)
```

Note: If you're using upstream sources, package versions in the upstream source that haven't yet been saved into your feed (by using them at least once) won't appear in the NuGet Package Manager search.

4. To install the packages and get it listed in our Custom Feed of DevOps Project
    a. Go to **https://www.nuget.org (one of the upstream sources)**
    b. Search the required Package and copy the **Install-Package** command.
        Example: **Install-Package Newtonsoft.Json -Version 12.0.3**
    c. In Visual Studio, open the Package Manager Console from **Tools → NuGet Package Manager**.
    d. Paste and run the command: **Install-Package Newtonsoft.Json -Version 12.0.3**.

5. Download and install packages from the upstream sources:

```
nuget restore
dotnet restore
```

6. Check your feed to see the saved copy of everything you used from the public registry

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

7. Now we can add Newtonsoft package reference to our project from the Custom Feed instead of Nuget.org

Views

- Views enable you to share **subsets** of the NuGet, npm, Maven, and Python package-versions in your feed with consumers.
- A common use for views is to share package versions that have been tested, validated, or deployed but hold back packages still under development and packages that didn't meet a quality bar.
- All Azure DevOps Services feeds come with 3 views**: @local, @prerelease, and @release**. The latter two are suggested views that you can rename or delete as desired. The @local view is a special view that's commonly used in upstream sources.
- Views are read-only, which means that users connected to a view can only use packages that are published to the feed and packages previously saved from upstream sources by users connected to the feed.

1. Go to Feed → Settings (Gear Icon on right) → Views Tab
2. **Edit a View**
    a. You can edit the name of Prerelease and Release but not Local
    b. You can restrict visibility to either all people in organization or specific people
3. **Promoting to a release view**
    a. Select the package → Click on . . . →
    b. select **Promote**
4. **Consuming from a release view**
    a. Select **Connect to feed**
    b. Use the feed URL with View name

https://pkgs.dev.azure.com/DevOpsDemoInJan2020/HelloWorld/_packaging/MyDemoFeed%40release/nuget/v3/index.json

Publish NuGet packages from your build to NuGet feeds

You can publish NuGet packages from your build to NuGet feeds. You can publish these packages to:

- Azure Artifacts or the TFS Package Management service.
- Other NuGet services such as NuGet.org.
- Your internal NuGet repository.

Step1: To create a package, add the following snippet to your azure-pipelines.yml file.

```
variables:
```

```
  Major: '1'
  Minor: '0'
  Patch: '0'

steps:
- task: NuGetCommand@2
  inputs:
    command: pack
    versioningScheme: byPrereleaseNumber
    majorVersion: '$(Major)'
    minorVersion: '$(Minor)'
    patchVersion: '$(Patch)'
OR
- task: DotNetCoreCLI@2
  displayName: 'dotnet pack'
  inputs:
    command: pack
    versioningScheme: byBuildNumber
```

Step2: Publish Your Packages to Azure Artifacts feed

1. To publish to an Azure Artifacts feed, set the **Project Collection Build Service** identity to be a **Contributor** on the feed
2. Add the following snippet to your azure-pipelines.yml file.

```
steps:
- task: NuGetCommand@2
  displayName: 'NuGet push'
  inputs:
    command: push
    publishVstsFeed: '<feedName>'
    allowPackageConflicts: true
OR
- task: DotNetCoreCLI@2
  displayName: 'dotnet push'
  inputs:
    command: push
```

```
publishVstsFeed: '<feedName>'
```

Step3: To publish to an external NuGet feed:

1. create a service connection to point to that feed. **Project settings**, → **Service connections**, → **New service connection**. → **NuGet** option for the service connection.
2. To connect to the feed, fill in the feed URL and the API key or token.
3. Add the following snippet to your azure-pipelines.yml file.

```
- task: NuGetAuthenticate@0
  inputs:
    nuGetServiceConnections: '<Name of the NuGet service connection>'

- task: NuGetCommand@2
  inputs:
    command: push
    nuGetFeedType: external
    versioningScheme: byEnvVar
    versionEnvVar: <VersionVariableName>
```

Universal Packages

Share code easily by storing Apache Maven, npm, and NuGet packages together. You can store packages using Universal Packages, eliminating the need to store binaries in Git.

Universal Packages store one or more files together in a single unit that has a name and version. You can publish Universal Packages from the command line by using the Azure CLI.

Log in to Azure DevOps

```
az login
az devops configure --defaults organization=https://dev.azure.com/[your-organization]
project=ContosoWebApp
```

Publish a Universal Package

```
az artifacts universal publish --organization https://dev.azure.com/fabrikam --feed FabrikamFiber --name my-first-package --version 1.0.0 --description "Your description" --path .
```

Download a Universal Package

```
az artifacts universal download --organization https://dev.azure.com/fabrikam --feed FabrikamFiber --name my-first-package --version 1.0.0 --path .
```

Filtered Universal Package downloads

```
az artifacts universal download --organization https://dev.azure.com/fabrikam --feed FabrikamFiber --name my-first-package --version 1.0.0 --path .  --file-filter **/*.exe;**/*.dll
```


Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)