

# AZ-400.4

## Module 1: Designing a Dependency Management Strategy



# Learning Objectives

- Recommend artifact management tools and practices
- Abstract common packages to enable sharing and reuse
- Inspect codebase to identify code dependencies that can be converted to packages
- Identify and recommend standardized package types and versions across the solution
- Refactor existing build pipelines to implement version strategy that publishes packages

# Lesson 01: Introduction



# Dependencies in software

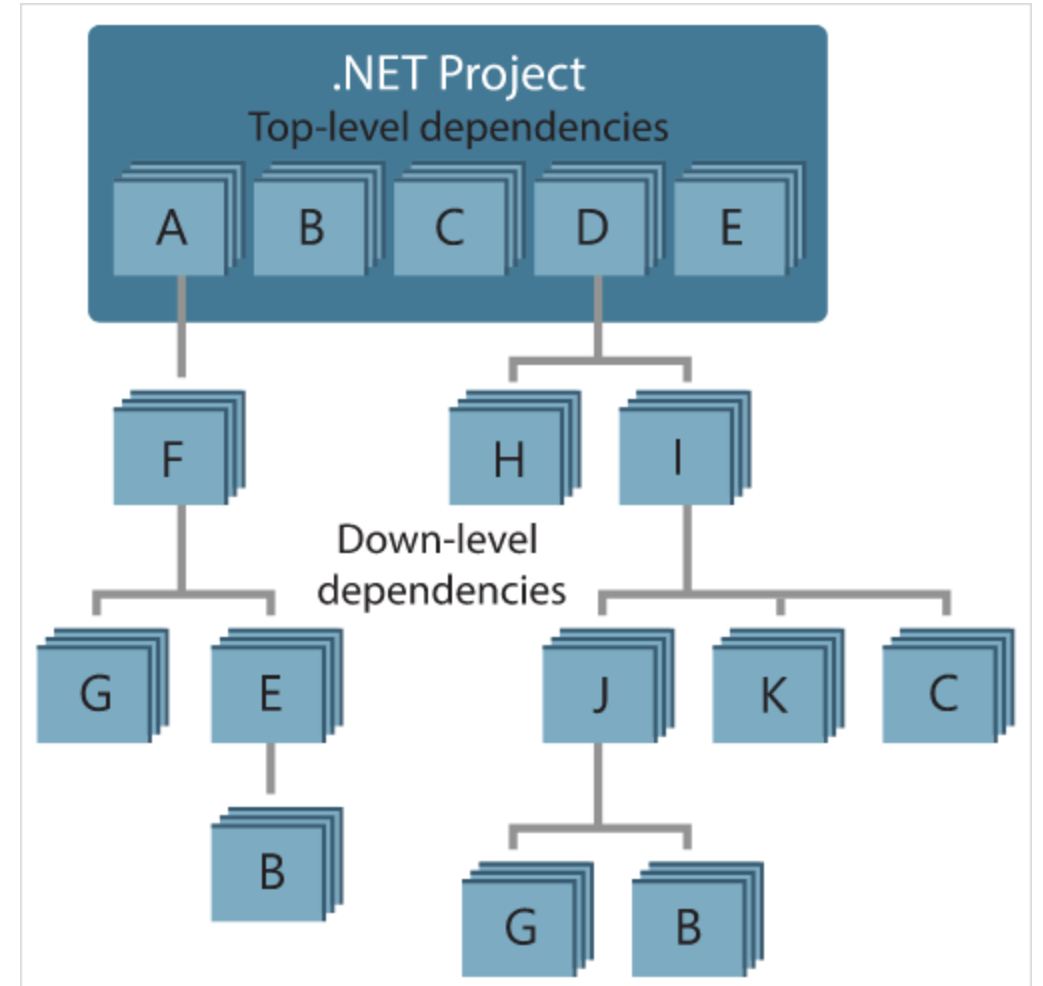
- Modern software is complex
- Component based development is common
- Not all software is written by a single team
- Dependencies on components created by other teams or persons

# Elements of a dependency management strategy

- There are a number of aspects for a dependency management strategy:
- Standardization
- Package formats and sources
- Versioning

# Decompose your system

1. Draw dependency graph
  2. Group components in sets of related components
- Few spanning check-ins across sets
  - Ideally a single team is responsible
  - Shared release cadence for single set



# Identifying dependencies

- Find components and source code that can have independent
  - Deployment
  - Release
  - Versioning
- Things to consider
  - Change frequency
  - Changes should be unrelated to other parts of system
  - Can package exist by itself
  - Package should add value for others

# Scanning your codebase for dependencies

- Duplicate code
- High cohesion and low coupling
- Individual lifecycle
- Stable parts
- Independent code and components



# Lesson 02: Packaging Dependencies



# What is a package?

- Mechanism to create, share and consume code and components
- Contains (compiled) code with metadata content for consuming packages

# Types of packages



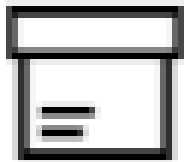
Microsoft platform and  
.NET artifacts



Node.js modules



Python scripts



Universal packages



Java packages



Docker images

- NuGet                      Microsoft development platform and .NET artifacts
- Npm                        node.js modules
- Maven                     Java packages
- Python                    Python script
- Universal                        Set of related files
- Docker                     Container images
- Symbols                        Symbol files

# Package feeds

- Centralized storage of package artifacts
  - Public or privately available
  - Offer secure access for private feeds
  - Versioned storage of packages
  - Managed by tooling
- Also known as
  - Package repositories
  - Package registry

# Package feed managers

- Manages feed
- Search and list packages from feed
- Consume packages
- Maintain local installation cache
- Publish packages
  
- Choose tooling:
  - Command-line tooling
  - Integrated in build and release pipelines

# Public sources

NuGet Gallery

<https://nuget.org>

NPMjs

<https://npmjs.org>

Maven

<https://search.maven.org>

Docker Hub

<https://hub.docker.com>

Python Package Index

<https://pypi.org>

# Self-hosted and SaaS based private package feeds

Package type	Self-hosted private feed	SaaS private feed
NuGet	NuGet server	Azure Artifacts, MyGet
NPM	Sinopia, <a href="https://cnpmjs.org/">cnpmjs.org</a> , Verdaccio	<a href="https://www.npmjs.com/">NPMjs.org</a> , 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



# Consuming packages

1. Identify a required dependency in your codebase
2. Find a component that satisfies the requirements for the project
3. Search the package sources for a package offering a correct version of the component
4. Install the package into the codebase and development machine
5. Create the software implementation that uses the new components from the package.

# Lesson 03: Package Management



# Azure DevOps



Azure Pipelines



Azure Boards



Azure Test Plans



Azure Repos



Azure Artifacts

# Azure Artifacts



Part of Azure DevOps

- Create private and public package feeds

Types of packages supported

1. NuGet
2. NPM
3. Maven
4. Universal
5. Python

# Creating a feed

- From Azure DevOps portal
- Feeds are centralized
- Specify
  - Name
  - Visibility
  - Public sources as upstreams



## Create new feed



Feeds host and control permissions for your packages.

Name \*

Team project - [\(what's this?\)](#)

DevOpsCertification-Course-MS

Visibility - Who can use your feed

- ☒  People in xpirit - Members of your organization can view the packages in your feed
- ☐  Specific people - Only people you give access to will be able to view this feed

Packages from public sources (nuget.org, npmjs.com)

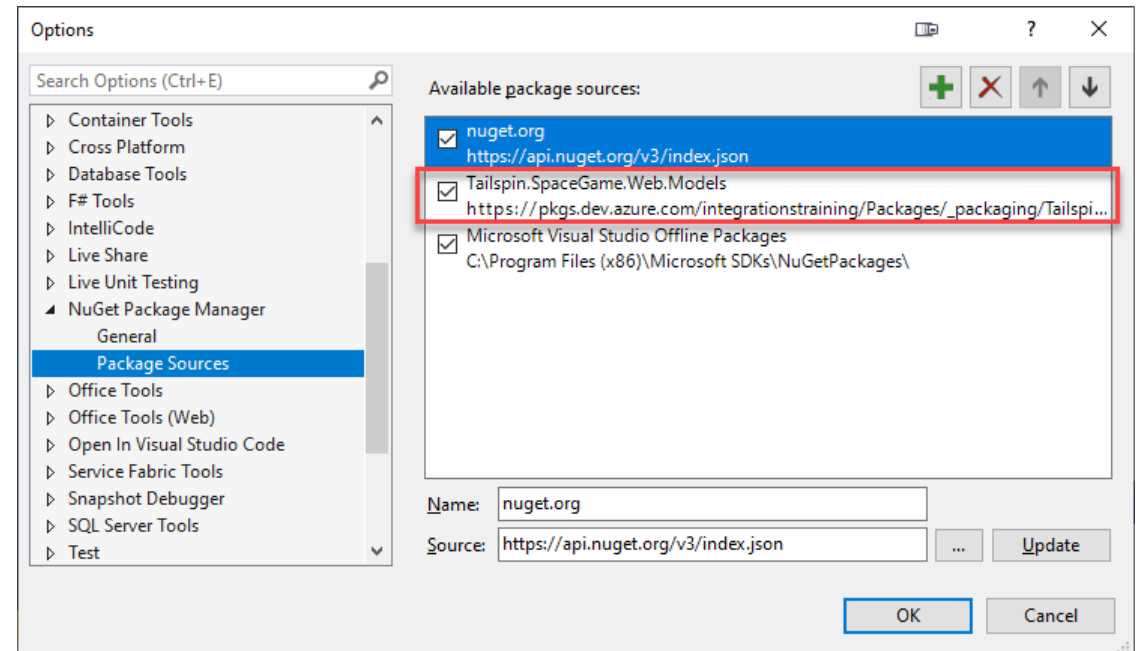
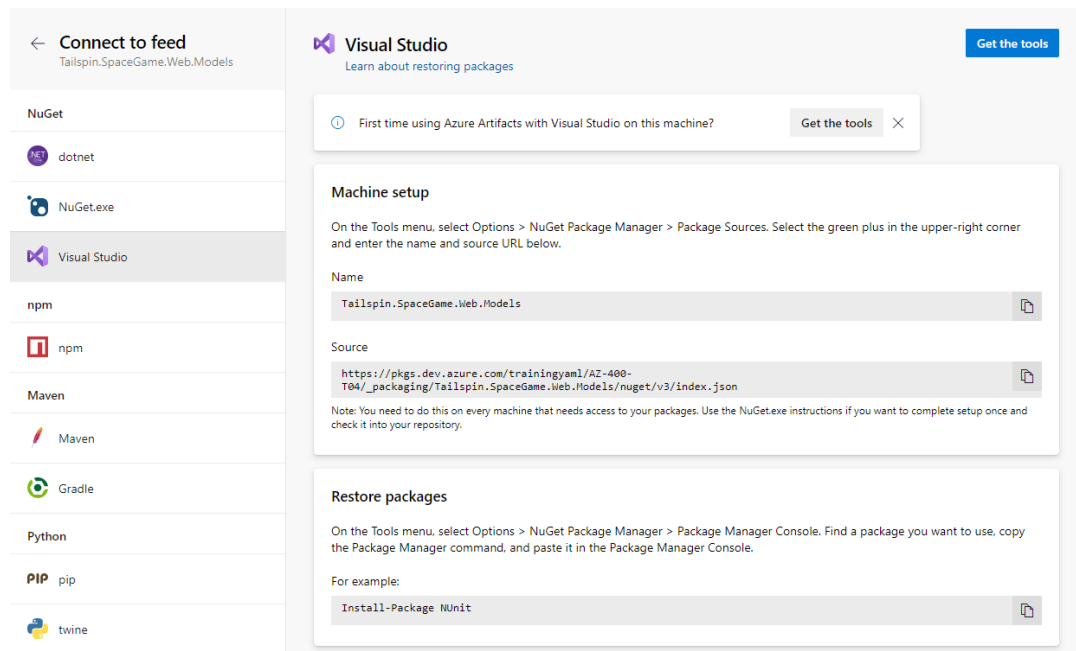
- ☒ Use packages from public sources through this feed
- ☐ Only use packages published to this feed

Create

Cancel

# Using a Feed in Visual Studio

- Got to Connect to Feed



# Creating a Package Feed



# Pushing a Package





# Promoting a package



# Package management

- Create a package feed
- Connect to the feed
- Create a NuGet package and publish it to the feed
- Import the new NuGet package into an existing project
- Update a NuGet package in the feed



# Pushing from the Pipeline




# Integrating in build pipelines

Why integrate in build pipeline?

- Automate to avoid errors
- Quality checks
- Implement a versioning strategy



Agent job 1

 Run on agent



NuGet restore

NuGet



Build solution

Visual Studio Build



Test Assemblies

 Visual Studio Test



WhiteSource Bolt

WhiteSource Bolt



Copy Files

Copy Files



Publish Artifact

Publish Build Artifacts

# Azure Pipelines tasks

- Different tasks for each of the package types
- Native integration with Azure Artifacts feeds
- Connecting with remote packages sources possible
- Requires authentication



## NuGet

Restore, pack, or push NuGet packages, or run a NuGet command. Supports NuGet.org and authenticated feeds like Package Management and MyGet. Uses NuGet.exe and works with .NET Framework apps. For .NET Core and .NET Standard apps, use the .NET Core task.



## NuGet Tool Installer

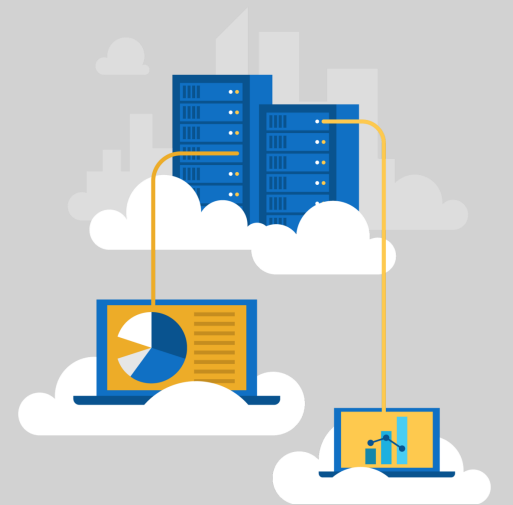
Acquires a specific version of NuGet from the internet or the tools cache and adds it to the PATH. Use this task to change the version of NuGet used in the NuGet tasks.



## .NET Core

Build, test, package, or publish a dotnet application, or run a custom dotnet command. For package commands, supports NuGet.org and authenticated feeds like Package Management and MyGet.

# Lesson 04: Implement a Versioning Strategy



# Introduction to versioning

Software changes - your code and dependencies alike

Versioning is about Compatibility!

Packages need to be versioned

- Identification
- Maintainability
- Each package has its own lifecycle and rate of change

# Immutable packages

Packages are immutable

- Once published a package cannot be changed
- Replacing or updating a package is not allowed
- Any change requires a new version



# Versioning of artifacts

- Way to express version technically varies per package type
- Versioning requires a scheme

Typical scheme:

2.1.15


Major Minor Patch

The diagram illustrates the semantic versioning scheme for the version number 2.1.15. The version is displayed in large black font. Below it, the words 'Major', 'Minor', and 'Patch' are positioned under the digits 2, 1, and 15 respectively. Light blue horizontal brackets connect each word to its corresponding part of the version number: 'Major' to '2', 'Minor' to '1', and 'Patch' to '15'.

# Semantic versioning

Express nature and risk of change

1.2.3-beta2



The diagram shows the version string '1.2.3-beta2' with two horizontal curly braces underneath. The first brace, in blue, is positioned under '1.2.3' and points to the text 'nature of change' below it. The second brace, in purple, is positioned under '-beta2' and points to the text 'quality of change' below it.

nature of  
change

quality of  
change

See also: <https://semver.org>

# Release views

Views help in defining quality without changing version numbers

Three default views:

1. Local
2. Release
3. Prerelease

```
https://pkgs.dev.azure.com/{yourteamproject}/_packaging/  
{feedname}@{Viewname}/nuget/v3/index.json
```

# Promoting packages


- Promote packages from @local view to other release views
- Upstream sources will only be evaluated from @local view
  - Only visible in other release views after being promoted

Promote this package

View

PartsUnlimited@Prerelease ▼

Promote Cancel

PartsUnlimited >  PartsUnlimited.Security 1.0.1

Overview

Versions



Connect to feed



Download



Promote



Unlist



Delete



Follow

Get this package



Connect to feed

then

PM> Install-Package PartsUnlimited.Security -version 1.0.1



# Promoting a Package



# AZ-400.3

## Module 2: Manage Security and Compliance



# Learning Objectives

- How to secure access to your packages and feeds
- Implications of using open source software
- Explain open source package licensing
- Integrating license and vulnerability scans
- Working with artifact repositories

# Lesson 01: Introduction





# Package security

Package feeds are trusted sources of packages

Feeds have to be secured:

- Private feeds
- Not allow access by unauthorized users for publishing

# Package compliance

Companies must be compliant to rules and regulations:

- Governmental
- Certification
- Standards

Open-source software have licenses that might violate compliancy rules

Compliancy should be guaranteed and provable

# Lesson 02: Package Security



# Securing access to package feeds

## Restricted access for consumption

Whenever a package feed and its packages should only be consumed by a certain audience, it is required to restrict access to it. Only those allowed access will be able to consume the packages from the feed

## Restricted access for publishing

Secure access is required to restrict who can publish so feeds and their packages cannot be modified by unauthorized or untrusted persons and accounts

# Roles

Available roles in Azure Artifacts:

- Reader:** Can list and restore (or install) packages from the feed
- Collaborator:** Is able to save packages from upstream sources
- Contributor:** Can push and unlist packages in the feed
- Owner:** has all available permissions for a package feed

Project Collection Build Service is contributor by default

DevOpsCertificationFeed > Feed settings

Feed detailsPermissionsViewsUpstream sources+ Add users/groupsDelete...

Filter by User/Group

User/Group	Role
Alex Thissen	Owner
[xpirit]\Project Collection Administrators	Owner
[DevOpsCertification-Course-MS]\Project Administrators	Owner
Project Collection Build Service (xpirit)	Contributor
[DevOpsCertification-Course-MS]\Contributors	Contributor

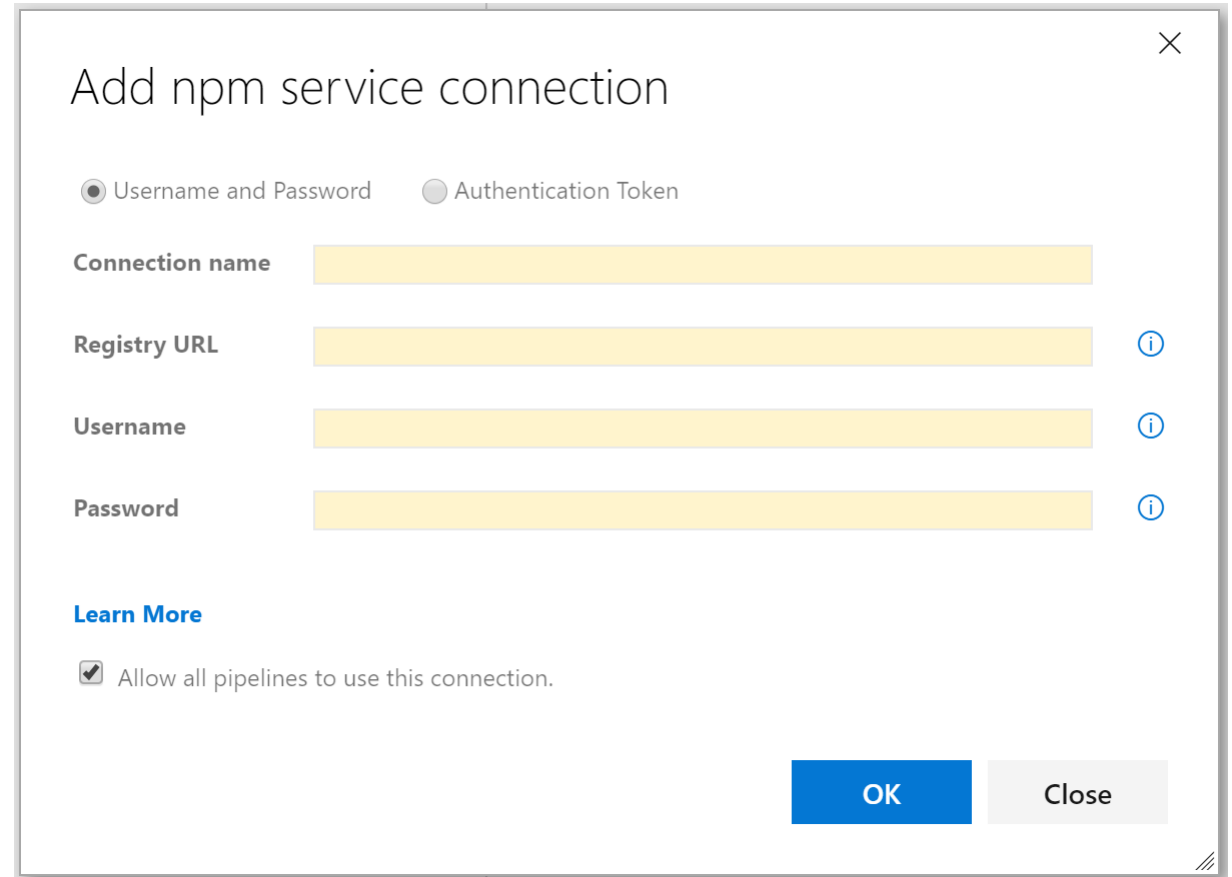
# Permissions

- Roles have certain permissions

Permission	Reader	Collaborator	Contributor	Owner
List and restore/install packages	✓	✓	✓	✓
Save packages from upstream sources		✓	✓	✓
Push packages			✓	✓
Unlist/deprecate packages			✓	✓
Delete/unpublish package				✓
Edit feed permissions				✓
Rename and delete feed				✓

# Credentials

- Authentication is required for Azure Artifacts
- Transparently taken care of when logged into portal or in build tasks
- External package sources may require credentials
  - Create a service connection



The screenshot shows a dialog box titled "Add npm service connection" with a close button (X) in the top right corner. Below the title, there are two radio buttons: "Username and Password" (selected) and "Authentication Token". The form contains four input fields: "Connection name", "Registry URL", "Username", and "Password". Each input field has a yellow background and a blue information icon (i) to its right. Below the input fields, there is a blue link labeled "Learn More". At the bottom, there is a checkbox labeled "Allow all pipelines to use this connection." which is checked. In the bottom right corner, there are two buttons: "OK" (blue) and "Close" (gray).

Add npm service connection

☒ Username and Password ☐ Authentication Token

Connection name

Registry URL  ⓘ

Username  ⓘ

Password  ⓘ

[Learn More](#)

☒ Allow all pipelines to use this connection.

OK Close

# Lesson 03: Open Source Software





# How software is built

Software based for 80% on components:

- Internal teams
- Commercial 3<sup>rd</sup> party
- Open-source community

Almost all software nowadays uses

open-source software in some way, shape or form

# What is open-source software?

“Open-source software (OSS) is a type of computer software in which source code is released under a license in which the copyright holder grants users the rights to study, change, and distribute the software to anyone and for any purpose.”

[https://en.wikipedia.org/wiki/Open-source\\_software](https://en.wikipedia.org/wiki/Open-source_software)

# Challenge to corporates

## Be of low quality

This would impact maintainability, reliability and performance of the overall solution

## Have no active maintenance

The code would not evolve over time, or be alterable without making a copy of the source code, effectively forking away from the origin.

## Contain malicious code

The entire system that includes and uses the code will be compromised. Potentially the entire company's IT and infrastructure is affected.

## Have security vulnerabilities

The security of a software system is as good as its weakest part. Using source code with vulnerabilities makes the entire system susceptible to attack by hackers and misuse.

## Have unfavorable licensing restrictions

The effect of a license can affect the entire solution that uses the open-source software.

# Open-source licenses

According to the open source definition of [OpenSource.org](https://opensource.org/) a license should not:

- Discriminate against persons or groups
- Discriminate against fields of endeavor
- Be specific to a product
- Restrict other software

# Open-source package license



# License implications and rating

Using a package implies following license requirements

Depending on license type this may have a high, medium or low impact on distributed software using it

License rating indicates impact of use of packages

- Compliancy
- Intellectual property
- Exclusive rights