Microsoft

AZ-400.00 Learning Path 08: Design and implement a dependency management strategy

Agenda

- Module 01: Explore package dependencies.
- Module 02: Understand package management.
- Module 03: Migrate, consolidating and secure artifacts.
- Module 04: Implement a versioning strategy.
- Module 05: Introduction to GitHub Packages.
- Labs & Learning Path review and takeaways.

Learning Path overview

Learning objectives

After completing this Learning Path, students will be able to:

1. Recommend artifact management tools and practices
2. Abstract common packages to enable sharing and reuse
3. Migrate and consolidate artifacts
4. Migrate and integrate source control measures

Module 01: Explore package dependencies

What is dependency management?

1. Modern software is complex
2. Component based development is common
3. Not all software is written by a single team
4. Dependencies on components created by other teams or persons

Describe elements of a dependency management strategy

- Standardization
- Package formats and sources
- Versioning

Identify 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

Understand source and package componentization

1. **Source componentization:**
   - Split out components
   - Related projects in different solutions

2. **Package componentization:**
   - Composing your solution to use packages

Decompose your system

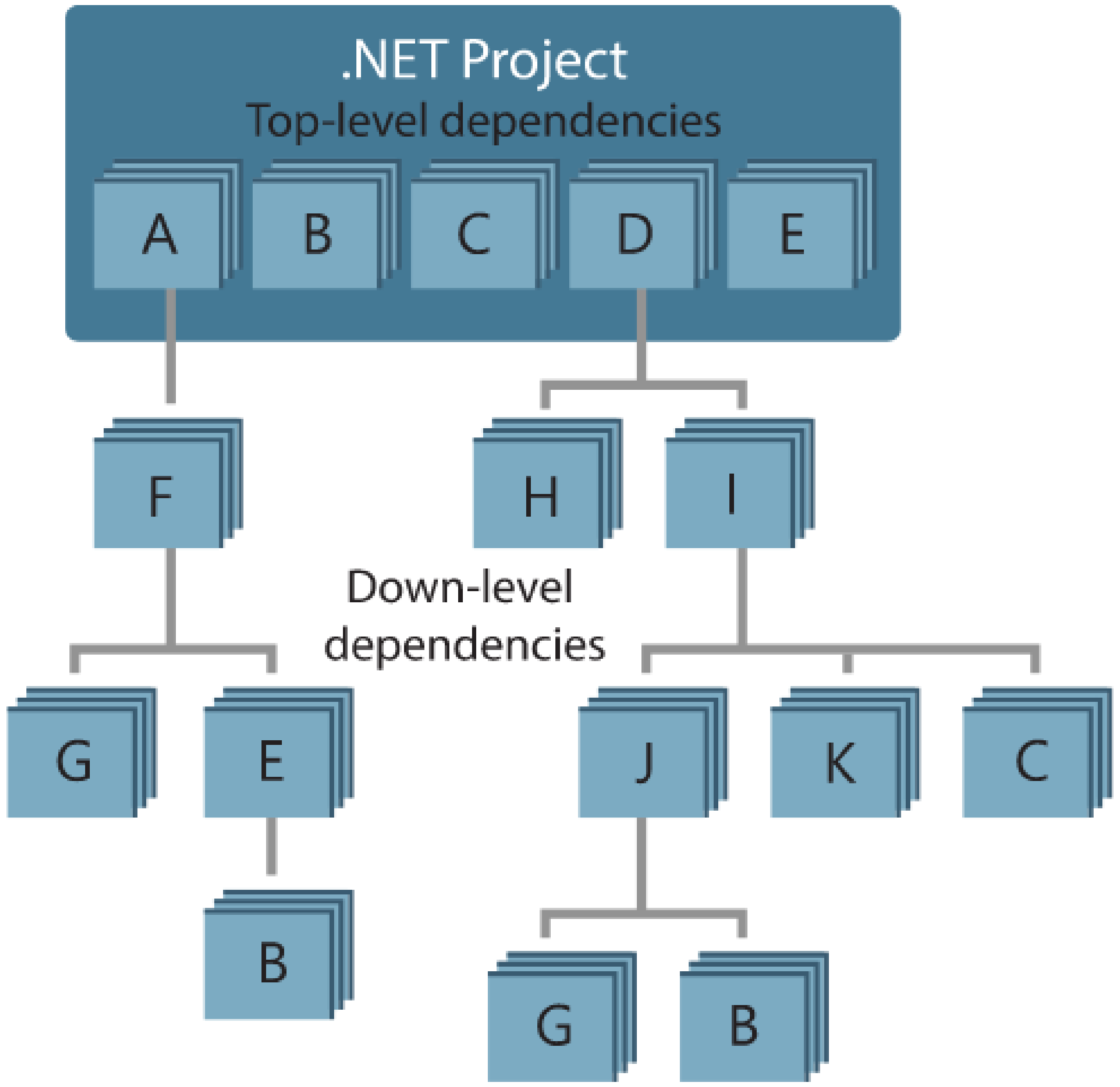
Approach:

1. Draw a 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



Scan your codebase for dependencies

1. Duplicate code
2. High cohesion and low coupling
3. Individual Lifecycle
4. Stable parts
5. Independent code and components

Module 02: Understand package management

Explore packages

A package is a formalized way of creating a distributable unit of software artifacts that can be consumed from another software solution.

Microsoft platform and .NET artifacts	Node.js modules	Python scripts
Universal packages	Java packages	Docker images

Understand 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 types:

Public: NuGet.org, Npmjs.org, PyPi.org, Docker Hub

Private: MyGet, Azure Container Registry, Azure Artifacts, Self-hosted solutions

Explore package feed managers

1. Manage feeds
2. Search and list packages from feed
3. Consume packages
4. Maintain local installation cache
5. Publish packages
6. **Choose tooling:**
    - Command-line tooling
    - Integrated in build and release pipelines

Explore common public package 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

Explore self-hosted and SaaS based package sources

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

Consume 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

Introduction to Azure Artifacts

Create private and public package feeds for package types:

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

Azure Artifacts

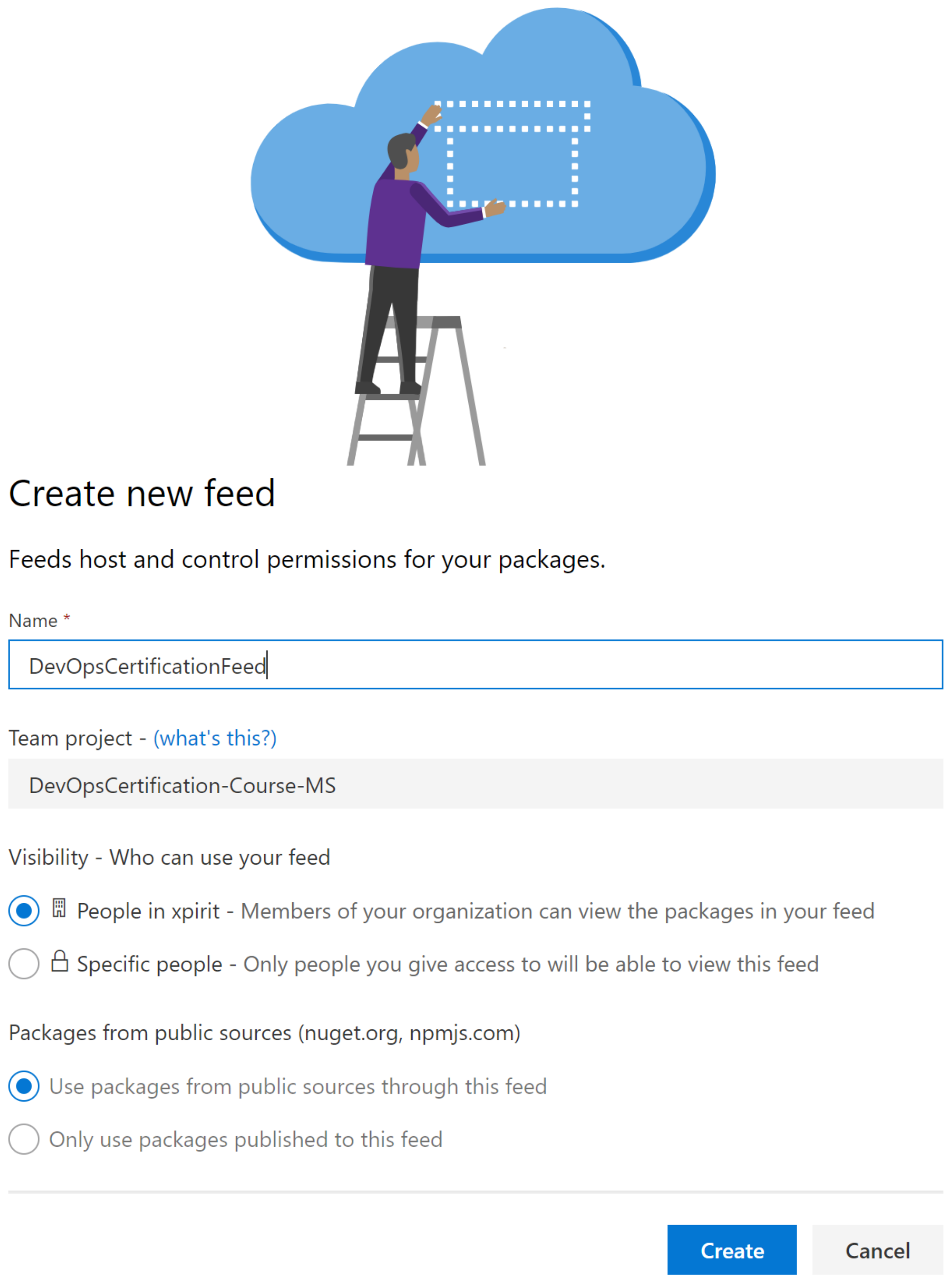
Publish packages

From Azure DevOps portal

Feeds are centralized

Specify:

- Name
- Visibility
- Public sources as upstream



Create new feed

Feeds host and control permissions for your packages.

Name *

DevOpsCertificationFeed

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

Demonstration: Create a package feed

DEMO

Demonstration: Push a package

DEMO

Module 03: Migrate, consolidating and secure artifacts

Identify existing artifact repositories

1. An artifact is a deployable component of your application.
2. Azure Pipelines can work with a wide variety of artifact sources and repositories.
3. Each release can specify which version of the artifacts are required.
4. Azure Artifacts can eliminate the need to manage file shares or to host private package servers.
5. Azure Artifacts provides universal artifact management for Maven, npm and NuGet.

Migrate and integrating artifact repositories

1. Get started with NuGet packages in Azure DevOps Services and TFS
2. Use npm to store JavaScript packages in Azure DevOps Services or TFS
3. Get started with Maven packages in Azure DevOps Services and TFS
4. Get started with Python packages in Azure Artifacts
5. Publish and then download a Universal Package

Secure access to package feeds

Feeds must be secured:

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

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.

Examine roles

Available roles in Azure Artifacts:

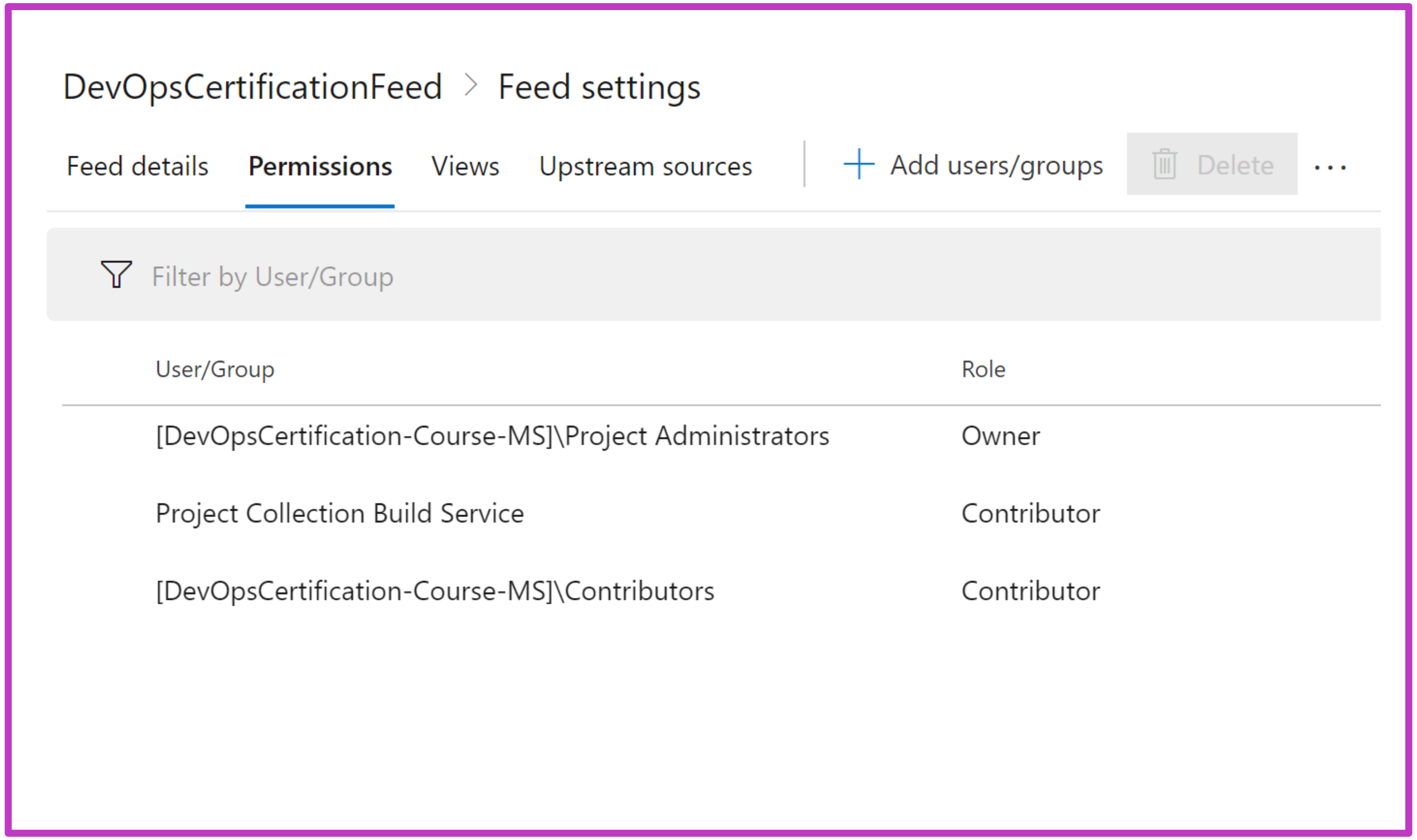
Reader: Can list and restore (or install) packages from the feed

Collaborator: Can 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



User/Group	Role
[DevOpsCertification-Course-MS]\Project Administrators	Owner
Project Collection Build Service	Contributor
[DevOpsCertification-Course-MS]\Contributors	Contributor

Examine permissions

Roles have certain permissions

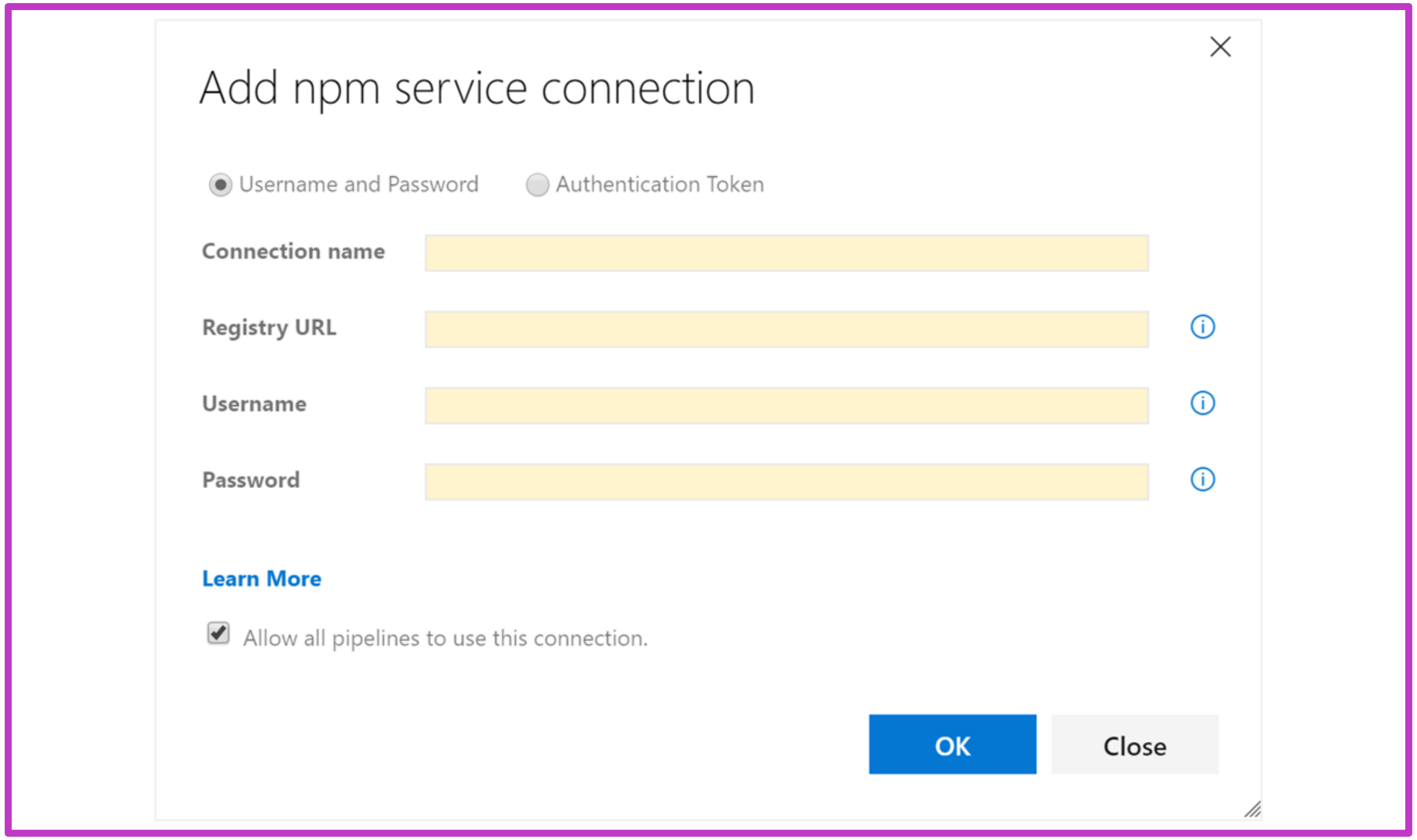
Permission	Reader	Collaborator	Contributor	Owner
List and restore/install packages	✓	✓	✓	✓
Save packages from upstream sources		✓	✓	✓
Push packages			✓	✓
Unlist/deprecate packages			✓	✓
Delete/unpublish package				✓
Edit feed permission				✓
Rename and delete feed				✓

Examine authentication

- 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



Add npm service connection

(● Username and Password) ○ Authentication Token

Connection name	
Registry URL	
Username	
Password	

Learn More

☑ Allow all pipelines to use this connection.

Module 04: Implement a versioning strategy

Introduction to versioning

1 Packages need to be versioned

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

2 Packages are immutable

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

Understand versioning of artifacts

Way to express version technically varies per package type

Versioning requires a scheme

Typical Scheme:

$$\underbrace{2.}_{\text{Major}}\underbrace{1.}_{\text{Minor}}\underbrace{15}_{\text{Patch}}$$

Explore semantic versioning

Express nature and risk of change

$$\underbrace{1.2.3}_{\text{Nature of change}}\underbrace{\text{-beta2}}_{\text{Quality of change}}$$

See also: https://semver.org

Examine release views

Views help in defining quality without changing version numbers

Three default views:

1. Local
2. Prerelease
3. Release

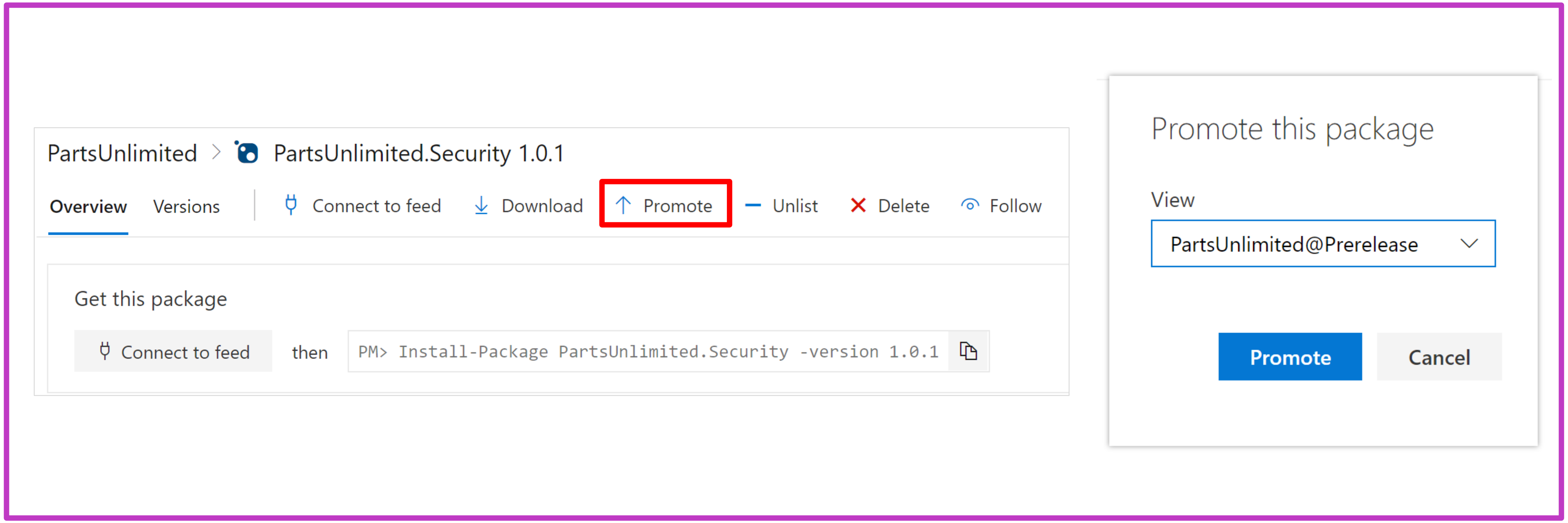
```
https://pkgs.dev.azure.com/{org}/{yourteamproject}/_packaging/{feedna
me}
@{Viewname}/nuget/v3/index.json
```

Promote 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



PartsUnlimited > PartsUnlimited.Security 1.0.1

Get this package

```
PM> Install-Package PartsUnlimited.Security -version 1.0.1
```

Promote this package

View: PartsUnlimited@Prerelease

Demonstration: Promote a package

DEMO

Explore best practices for versioning

1. Have a documented versioning strategy
2. Adopt SemVer 2.0 for your versioning scheme
3. Each repository should only reference one feed
4. On package creation, automatically publish packages back to the feed

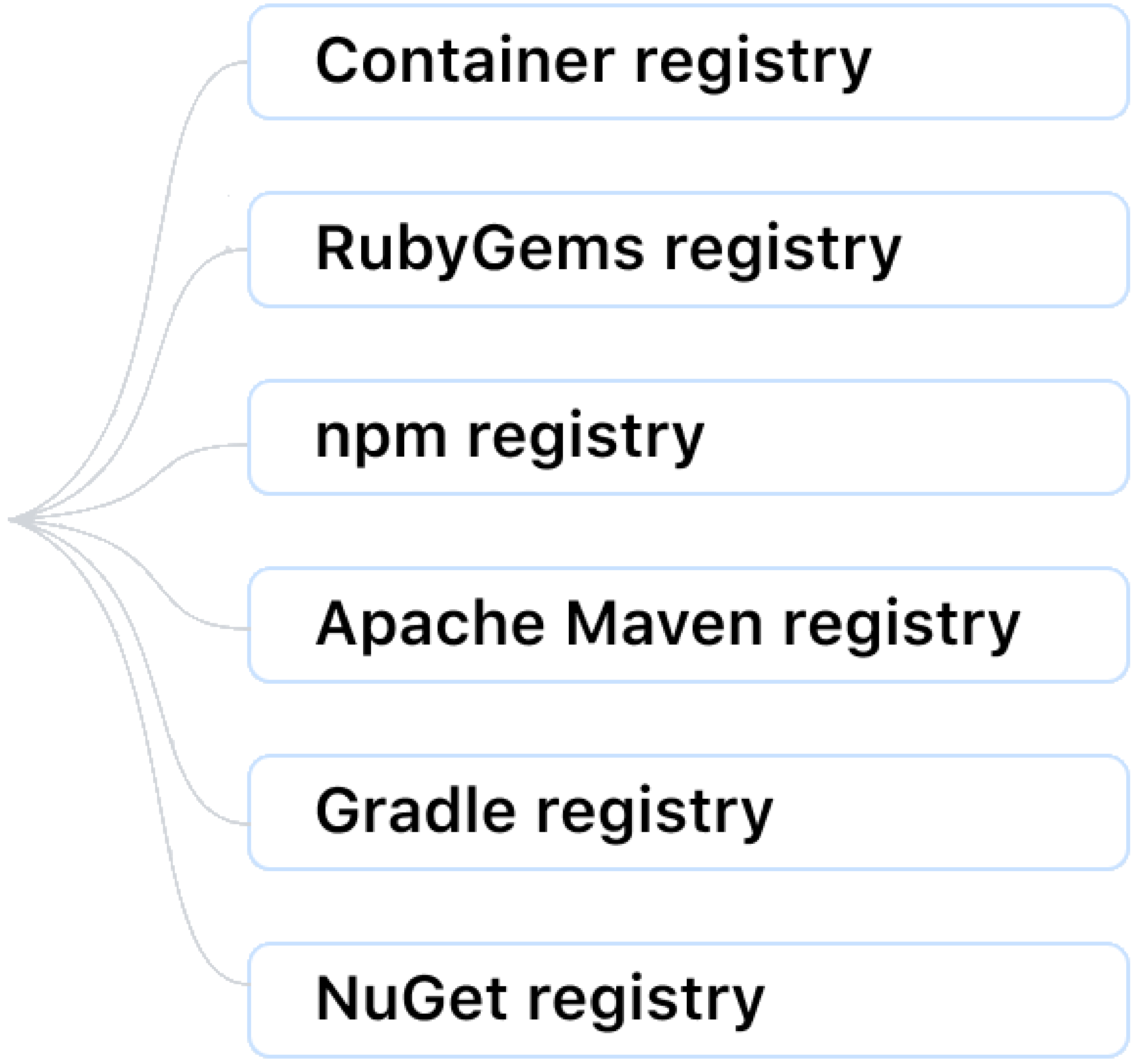
Demonstration: Push from the pipeline

DEMO

Module 05: Introduction to GitHub Packages

Introduction to GitHub Packages

- **GitHub Packages is a platform for hosting and managing packages**
  - Combine source code and packages in one place.
  - Integrate permissions management and billing.
  - Integrate GitHub Packages with GitHub APIs, GitHub Actions, and webhooks.



Publish packages

GitHub Packages use native package tooling commands to publish and install package versions.

Create your token, scope, authenticate and publish.

Language	Package format	Package client
JavaScript	package.json	npm
Ruby	Gemfile	gem
Java	pom.xml	mvn
Java	build.gradle or build.gradle.kts	gradle
.NET	nupkg	dotnet CLI

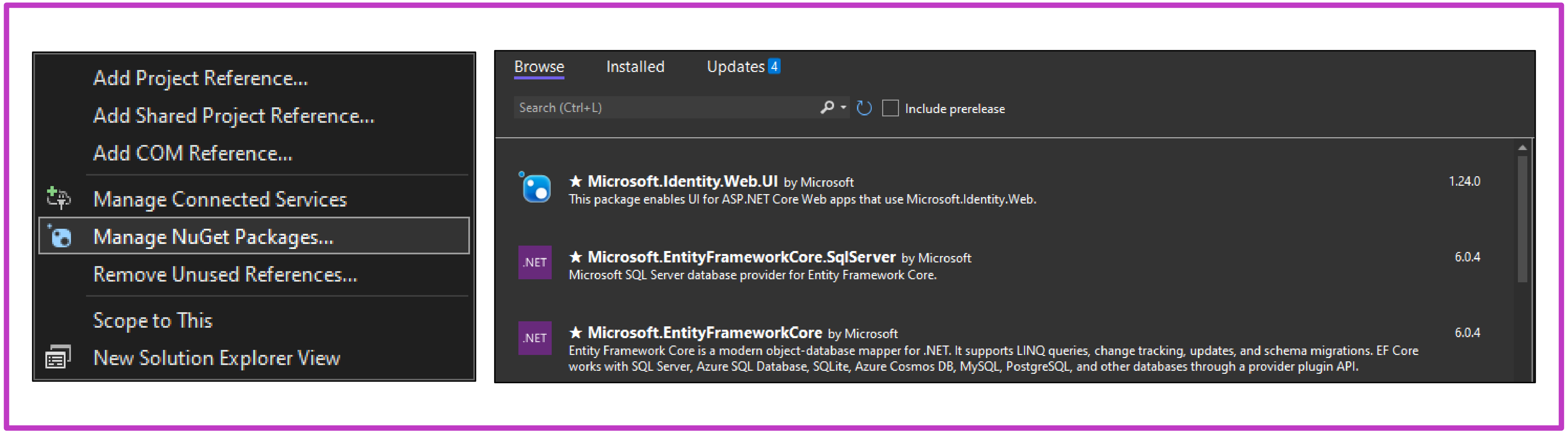
Publish packages – nuget.config

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <packageSources>
        <clear />
        <add key="github" value="https://nuget.pkg.github.com/OWNER/index.json" />
    </packageSources>
    <packageSourceCredentials>
        <github>
            <add key="Username" value="USERNAME" />
            <add key="ClearTextPassword" value="TOKEN" />
        </github>
    </packageSourceCredentials>
</configuration>
```

Install a package

You can install a package using any supported package client following the same general guidelines:

- Authenticate to GitHub Packages using the instructions for your package client.
- Install the package using the instructions for your package client.



★ **Microsoft.Identity.Web.UI** by Microsoft 1.24.0
This package enables UI for ASP.NET Core Web apps that use Microsoft.Identity.Web.

★ **Microsoft.EntityFrameworkCore.SqlServer** by Microsoft 6.0.4
Microsoft SQL Server database provider for Entity Framework Core.

★ **Microsoft.EntityFrameworkCore** by Microsoft 6.0.4
Entity Framework Core is a modern object-database mapper for .NET. It supports LINQ queries, change tracking, updates, and schema migrations. EF Core works with SQL Server, Azure SQL Database, SQLite, Azure Cosmos DB, MySQL, PostgreSQL, and other databases through a provider plugin API.

Delete and restore a package

1. **You can delete it on GitHub if you have the required access:**
   - An entire private package.
   - An entire public package, if there are not more than 5000 downloads of any version of the package.
   - A specific version of a private package.
   - A specific version of a public package, if the package version doesn't have more than 5000 downloads.
2. **You can also restore an entire package or package version, if:**
   - You restore the package within 30 days of its deletion.
   - The same package namespace is still available and not used for a new package.

Explore package access control and visibility

Visibility and access permissions for container images:

Read:

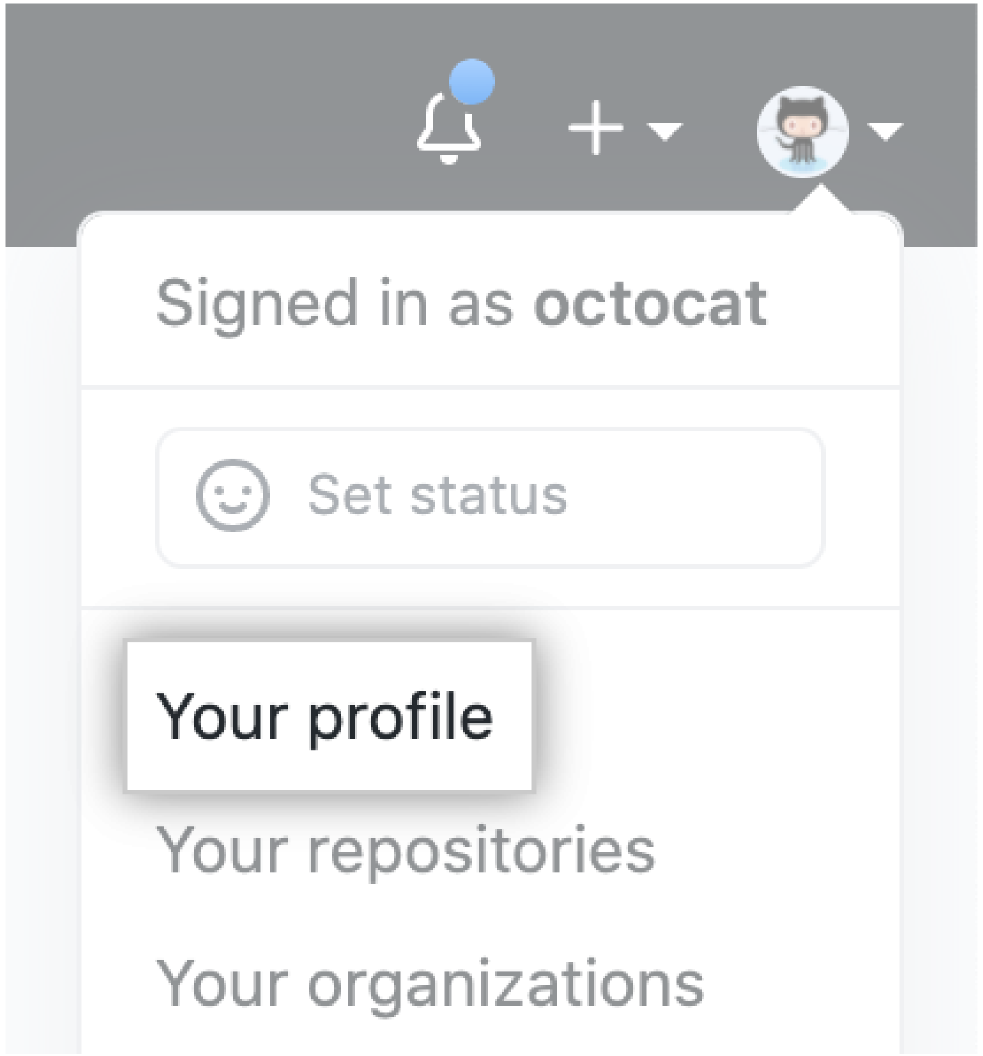
- Can download package.
- Can read package metadata.

Write:

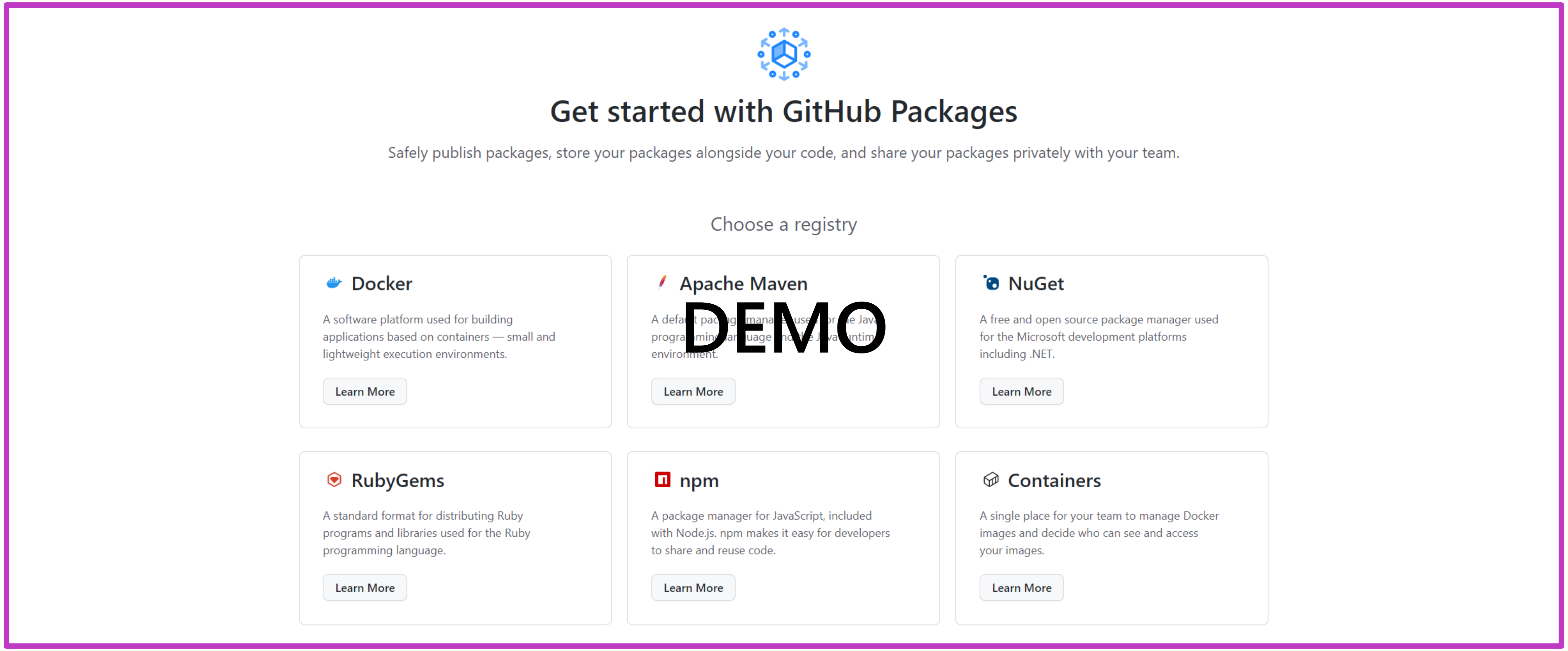
- Can upload and download this package.
- Can read and write package metadata.

Admin:

- Can upload, download, delete, and manage this package.
- Can read and write package metadata.
- Can grant package permissions.



Demonstration: Install, delete and restore packages using GitHub



Get started with GitHub Packages

Safely publish packages, store your packages alongside your code, and share your packages privately with your team.

Choose a registry

Docker

A software platform used for building applications based on containers — small and lightweight execution environments.

Learn More

Apache Maven

A default package manager used for the Java programming language and the Java runtime environment.

Learn More

DEMO

NuGet

A free and open source package manager used for the Microsoft development platforms including .NET.

Learn More

RubyGems

A standard format for distributing Ruby programs and libraries used for the Ruby programming language.

Learn More

npm

A package manager for JavaScript, included with Node.js. npm makes it easy for developers to share and reuse code.

Learn More

Containers

A single place for your team to manage Docker images and decide who can see and access your images.

Learn More

Labs

Lab: Package management with Azure Artifacts

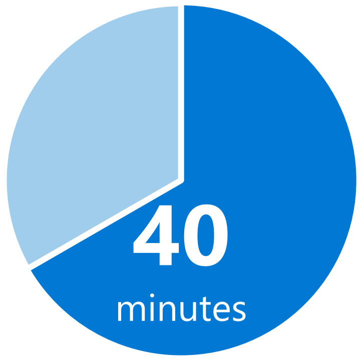
Lab overview:

In this lab, you will learn how to work with Azure Artifacts.

Objectives:

- Create and connect to a feed.
- Create and publish a NuGet package.
- Import a NuGet package.
- Update a NuGet package.

Duration:



Learning Path review and takeaways

What did you learn?

1. Recommend artifact management tools and practices
2. Abstract common packages to enable sharing and reuse
3. Migrate and consolidate artifacts
4. Migrate and integrate source control measures

Learning Path review questions

1. If you are creating a feed that will allow yourself and those that you invite to publish, what visibility should you choose?
2. Can you create a package feed for Maven in Azure Artifacts?
3. What type of package should you use for Machine learning training data & models?
4. If an existing package is found to be broken or buggy, how should it be fixed?
5. What is meant by saying that a package should be immutable?

Microsoft