

Microsoft Official Course



AZ-400T04

Implementing Dependency Management

AZ-400T04

Implementing Dependency Management

Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

The names of manufacturers, products, or URLs are provided for informational purposes only and Microsoft makes no representations and warranties, either expressed, implied, or statutory, regarding these manufacturers or the use of the products with any Microsoft technologies. The inclusion of a manufacturer or product does not imply endorsement of Microsoft of the manufacturer or product. Links may be provided to third party sites. Such sites are not under the control of Microsoft and Microsoft is not responsible for the contents of any linked site or any link contained in a linked site, or any changes or updates to such sites. Microsoft is not responsible for webcasting or any other form of transmission received from any linked site. Microsoft is providing these links to you only as a convenience, and the inclusion of any link does not imply endorsement of Microsoft of the site or the products contained therein.

© 2019 Microsoft Corporation. All rights reserved.

Microsoft and the trademarks listed at <http://www.microsoft.com/trademarks>¹ are trademarks of the Microsoft group of companies. All other trademarks are property of their respective owners.

¹ <http://www.microsoft.com/trademarks>

MICROSOFT LICENSE TERMS

MICROSOFT INSTRUCTOR-LED COURSEWARE

These license terms are an agreement between Microsoft Corporation (or based on where you live, one of its affiliates) and you. Please read them. They apply to your use of the content accompanying this agreement which includes the media on which you received it, if any. These license terms also apply to Trainer Content and any updates and supplements for the Licensed Content unless other terms accompany those items. If so, those terms apply.

BY ACCESSING, DOWNLOADING OR USING THE LICENSED CONTENT, YOU ACCEPT THESE TERMS. IF YOU DO NOT ACCEPT THEM, DO NOT ACCESS, DOWNLOAD OR USE THE LICENSED CONTENT.

If you comply with these license terms, you have the rights below for each license you acquire.

1. DEFINITIONS.

- a. "Authorized Learning Center" means a Microsoft IT Academy Program Member, Microsoft Learning Competency Member, or such other entity as Microsoft may designate from time to time.
- b. "Authorized Training Session" means the instructor-led training class using Microsoft Instructor-Led Courseware conducted by a Trainer at or through an Authorized Learning Center.
- c. "Classroom Device" means one (1) dedicated, secure computer that an Authorized Learning Center owns or controls that is located at an Authorized Learning Center's training facilities that meets or exceeds the hardware level specified for the particular Microsoft Instructor-Led Courseware.
- d. "End User" means an individual who is (i) duly enrolled in and attending an Authorized Training Session or Private Training Session, (ii) an employee of a MPN Member, or (iii) a Microsoft full-time employee.
- e. "Licensed Content" means the content accompanying this agreement which may include the Microsoft Instructor-Led Courseware or Trainer Content.
- f. "Microsoft Certified Trainer" or "MCT" means an individual who is (i) engaged to teach a training session to End Users on behalf of an Authorized Learning Center or MPN Member, and (ii) currently certified as a Microsoft Certified Trainer under the Microsoft Certification Program.

g. "Microsoft Instructor-Led Courseware" means the Microsoft-branded instructor-led training course that educates IT professionals and developers on Microsoft technologies. A Microsoft Instructor-Led Courseware title may be branded as MOC, Microsoft Dynamics or Microsoft Business Group courseware.

h. "Microsoft IT Academy Program Member" means an active member of the Microsoft IT Academy Program.

i. "Microsoft Learning Competency Member" means an active member of the Microsoft Partner Network program in good standing that currently holds the Learning Competency status.

j. "MOC" means the "Official Microsoft Learning Product" instructor-led courseware known as Microsoft Official Course that educates IT professionals and developers on Microsoft technologies.

k. "MPN Member" means an active Microsoft Partner Network program member in good standing.

l. "Personal Device" means one (1) personal computer, device, workstation or other digital electronic device that you personally own or control that meets or exceeds the hardware level specified for the particular Microsoft Instructor-Led Courseware.

m. "Private Training Session" means the instructor-led training classes provided by MPN Members for corporate customers to teach a predefined learning objective using Microsoft Instructor-Led Courseware. These classes are not advertised or promoted to the general public and class attendance is restricted to individuals employed by or contracted by the corporate customer.

n. "Trainer" means (i) an academically accredited educator engaged by a Microsoft IT Academy Program Member to teach an Authorized Training Session, and/or (ii) a MCT.

o. "Trainer Content" means the trainer version of the Microsoft Instructor-Led Courseware and additional supplemental content designated solely for Trainers' use to teach a training session using the Microsoft Instructor-Led Courseware. Trainer Content may include Microsoft PowerPoint presentations, trainer preparation guide, train the trainer materials, Microsoft One Note packs, classroom setup guide and Pre-release course feedback form. To clarify, Trainer Content does not include any software, virtual hard disks or virtual machines.

2. USE RIGHTS. The Licensed Content is licensed not sold. The Licensed Content is licensed on a one copy per user basis, such that you must acquire a license for each individual that accesses or uses the Licensed Content.

2.1 Below are five separate sets of use rights. Only one set of rights apply to you.

a. If you are a Microsoft IT Academy Program Member:

- i. Each license acquired on behalf of yourself may only be used to review one (1) copy of the Microsoft Instructor-Led Courseware in the form provided to you. If the Microsoft Instructor-Led Courseware is in digital format, you may install one (1) copy on up to three (3) Personal Devices. You may not install the Microsoft Instructor-Led Courseware on a device you do not own or control.
- ii. For each license you acquire on behalf of an End User or Trainer, you may either:
 - 1. distribute one (1) hard copy version of the Microsoft Instructor-Led Courseware to one (1) End User who is enrolled in the Authorized Training Session, and only immediately prior to the commencement of the Authorized Training Session that is the subject matter of the Microsoft Instructor-Led Courseware being provided, or
 - 2. provide one (1) End User with the unique redemption code and instructions on how they can access one (1) digital version of the Microsoft Instructor-Led Courseware, or
 - 3. provide one (1) Trainer with the unique redemption code and instructions on how they can access one (1) Trainer Content,provided you comply with the following:
 - iii. you will only provide access to the Licensed Content to those individuals who have acquired a valid license to the Licensed Content,
 - iv. you will ensure each End User attending an Authorized Training Session has their own valid licensed copy of the Microsoft Instructor-Led Courseware that is the subject of the Authorized Training Session,
 - v. you will ensure that each End User provided with the hard-copy version of the Microsoft Instructor-Led Courseware will be presented with a copy of this agreement and each End User will agree that their use of the Microsoft Instructor-Led Courseware will be subject to the terms in this agreement prior to providing them with the Microsoft Instructor-Led Courseware. Each individual will be required to denote their acceptance of this agreement in a manner that is enforceable under local law prior to their accessing the Microsoft Instructor-Led Courseware,
 - vi. you will ensure that each Trainer teaching an Authorized Training Session has their own valid licensed copy of the Trainer Content that is the subject of the Authorized Training Session,
 - vii. you will only use qualified Trainers who have in-depth knowledge of and experience with the Microsoft technology that is the subject of the Microsoft Instructor-Led Courseware being taught for all your Authorized Training Sessions,
 - viii. you will only deliver a maximum of 15 hours of training per week for each Authorized Training Session that uses a MOC title, and
 - ix. you acknowledge that Trainers that are not MCTs will not have access to all of the trainer resources for the Microsoft Instructor-Led Courseware.
- b. If you are a Microsoft Learning Competency Member:
 - i. Each license acquired on behalf of yourself may only be used to review one (1) copy of the Microsoft Instructor-Led Courseware in the form provided to you. If the Microsoft Instructor-Led Courseware is in digital format, you may install one (1) copy on up to three (3) Personal Devices. You may not install the Microsoft Instructor-Led Courseware on a device you do not own or control.
 - ii. For each license you acquire on behalf of an End User or MCT, you may either:
 - 1. distribute one (1) hard copy version of the Microsoft Instructor-Led Courseware to one (1) End User attending the Authorized Training Session and only immediately prior to the commencement of the Authorized Training Session that is the subject matter of the Microsoft Instructor-Led Courseware provided, or

2. provide one (1) End User attending the Authorized Training Session with the unique redemption code and instructions on how they can access one (1) digital version of the Microsoft Instructor-Led Courseware, or

3. you will provide one (1) MCT with the unique redemption code and instructions on how they can access one (1) Trainer Content,

provided you comply with the following:

iii. you will only provide access to the Licensed Content to those individuals who have acquired a valid license to the Licensed Content,

iv. you will ensure that each End User attending an Authorized Training Session has their own valid licensed copy of the Microsoft Instructor-Led Courseware that is the subject of the Authorized Training Session,

v. you will ensure that each End User provided with a hard-copy version of the Microsoft Instructor-Led Courseware will be presented with a copy of this agreement and each End User will agree that their use of the Microsoft Instructor-Led Courseware will be subject to the terms in this agreement prior to providing them with the Microsoft Instructor-Led Courseware. Each individual will be required to denote their acceptance of this agreement in a manner that is enforceable under local law prior to their accessing the Microsoft Instructor-Led Courseware,

vi. you will ensure that each MCT teaching an Authorized Training Session has their own valid licensed copy of the Trainer Content that is the subject of the Authorized Training Session,

vii. you will only use qualified MCTs who also hold the applicable Microsoft Certification credential that is the subject of the MOC title being taught for all your Authorized Training Sessions using MOC,

viii. you will only provide access to the Microsoft Instructor-Led Courseware to End Users, and

ix. you will only provide access to the Trainer Content to MCTs.

c. If you are a MPN Member:

i. Each license acquired on behalf of yourself may only be used to review one (1) copy of the Microsoft Instructor-Led Courseware in the form provided to you. If the Microsoft Instructor-Led Courseware is in digital format, you may install one (1) copy on up to three (3) Personal Devices. You may not install the Microsoft Instructor-Led Courseware on a device you do not own or control.

ii. For each license you acquire on behalf of an End User or Trainer, you may either:

1. distribute one (1) hard copy version of the Microsoft Instructor-Led Courseware to one (1) End User attending the Private Training Session, and only immediately prior to the commencement of the Private Training Session that is the subject matter of the Microsoft Instructor-Led Courseware being provided, or

2. provide one (1) End User who is attending the Private Training Session with the unique redemption code and instructions on how they can access one (1) digital version of the Microsoft Instructor-Led Courseware, or

3. you will provide one (1) Trainer who is teaching the Private Training Session with the unique redemption code and instructions on how they can access one (1) Trainer Content,

provided you comply with the following:

iii. you will only provide access to the Licensed Content to those individuals who have acquired a valid license to the Licensed Content,

iv. you will ensure that each End User attending an Private Training Session has their own valid licensed copy of the Microsoft Instructor-Led Courseware that is the subject of the Private Training Session,

- v. you will ensure that each End User provided with a hard copy version of the Microsoft Instructor-Led Courseware will be presented with a copy of this agreement and each End User will agree that their use of the Microsoft Instructor-Led Courseware will be subject to the terms in this agreement prior to providing them with the Microsoft Instructor-Led Courseware. Each individual will be required to denote their acceptance of this agreement in a manner that is enforceable under local law prior to their accessing the Microsoft Instructor-Led Courseware,
- vi. you will ensure that each Trainer teaching an Private Training Session has their own valid licensed copy of the Trainer Content that is the subject of the Private Training Session,
- vii. you will only use qualified Trainers who hold the applicable Microsoft Certification credential that is the subject of the Microsoft Instructor-Led Courseware being taught for all your Private Training Sessions,
- viii. you will only use qualified MCTs who hold the applicable Microsoft Certification credential that is the subject of the MOC title being taught for all your Private Training Sessions using MOC,
- ix. you will only provide access to the Microsoft Instructor-Led Courseware to End Users, and
- x. you will only provide access to the Trainer Content to Trainers.

d. If you are an End User:

For each license you acquire, you may use the Microsoft Instructor-Led Courseware solely for your personal training use. If the Microsoft Instructor-Led Courseware is in digital format, you may access the Microsoft Instructor-Led Courseware online using the unique redemption code provided to you by the training provider and install and use one (1) copy of the Microsoft Instructor-Led Courseware on up to three (3) Personal Devices. You may also print one (1) copy of the Microsoft Instructor-Led Courseware. You may not install the Microsoft Instructor-Led Courseware on a device you do not own or control.

e. If you are a Trainer.

i. For each license you acquire, you may install and use one (1) copy of the Trainer Content in the form provided to you on one (1) Personal Device solely to prepare and deliver an Authorized Training Session or Private Training Session, and install one (1) additional copy on another Personal Device as a backup copy, which may be used only to reinstall the Trainer Content. You may not install or use a copy of the Trainer Content on a device you do not own or control. You may also print one (1) copy of the Trainer Content solely to prepare for and deliver an Authorized Training Session or Private Training Session.

ii. You may customize the written portions of the Trainer Content that are logically associated with instruction of a training session in accordance with the most recent version of the MCT agreement. If you elect to exercise the foregoing rights, you agree to comply with the following: (i) customizations may only be used for teaching Authorized Training Sessions and Private Training Sessions, and (ii) all customizations will comply with this agreement. For clarity, any use of "customize" refers only to changing the order of slides and content, and/or not using all the slides or content, it does not mean changing or modifying any slide or content.

2.2 Separation of Components. The Licensed Content is licensed as a single unit and you may not separate their components and install them on different devices.

2.3 **Redistribution of Licensed Content.** Except as expressly provided in the use rights above, you may not distribute any Licensed Content or any portion thereof (including any permitted modifications) to any third parties without the express written permission of Microsoft.

2.4 **Third Party Notices.** The Licensed Content may include third party code that Microsoft, not the third party, licenses to you under this agreement. Notices, if any, for the third party code are included for your information only.

2.5 **Additional Terms.** Some Licensed Content may contain components with additional terms, conditions, and licenses regarding its use. Any non-conflicting terms in those conditions and licenses also apply to your use of that respective component and supplements the terms described in this agreement.

3. **LICENSED CONTENT BASED ON PRE-RELEASE TECHNOLOGY.** If the Licensed Content's subject matter is based on a pre-release version of Microsoft technology ("Pre-release"), then in addition to the other provisions in this agreement, these terms also apply:

a. **Pre-Release Licensed Content.** This Licensed Content subject matter is on the Pre-release version of the Microsoft technology. The technology may not work the way a final version of the technology will and we may change the technology for the final version. We also may not release a final version. Licensed Content based on the final version of the technology may not contain the same information as the Licensed Content based on the Pre-release version. Microsoft is under no obligation to provide you with any further content, including any Licensed Content based on the final version of the technology.

b. **Feedback.** If you agree to give feedback about the Licensed Content to Microsoft, either directly or through its third party designee, you give to Microsoft without charge, the right to use, share and commercialize your feedback in any way and for any purpose. You also give to third parties, without charge, any patent rights needed for their products, technologies and services to use or interface with any specific parts of a Microsoft technology, Microsoft product, or service that includes the feedback. You will not give feedback that is subject to a license that requires Microsoft to license its technology, technologies, or products to third parties because we include your feedback in them. These rights survive this agreement.

c. **Pre-release Term.** If you are an Microsoft IT Academy Program Member, Microsoft Learning Competency Member, MPN Member or Trainer, you will cease using all copies of the Licensed Content on the Pre-release technology upon (i) the date which Microsoft informs you is the end date for using the Licensed Content on the Pre-release technology, or (ii) sixty (60) days after the commercial release of the technology that is the subject of the Licensed Content, whichever is earliest ("Pre-release term"). Upon expiration or termination of the Pre-release term, you will irretrievably delete and destroy all copies of the Licensed Content in your possession or under your control.

4. **SCOPE OF LICENSE.** The Licensed Content is licensed, not sold. This agreement only gives you some rights to use the Licensed Content. Microsoft reserves all other rights. Unless applicable law gives you more rights despite this limitation, you may use the Licensed Content only as expressly permitted in this agreement. In doing so, you must comply with any technical limitations in the Licensed Content that only allows you to use it in certain ways. Except as expressly permitted in this agreement, you may not:

- access or allow any individual to access the Licensed Content if they have not acquired a valid license for the Licensed Content,
- alter, remove or obscure any copyright or other protective notices (including watermarks), branding or identifications contained in the Licensed Content,
- modify or create a derivative work of any Licensed Content,
- publicly display, or make the Licensed Content available for others to access or use,
- copy, print, install, sell, publish, transmit, lend, adapt, reuse, link to or post, make available or distribute the Licensed Content to any third party,
- work around any technical limitations in the Licensed Content, or
- reverse engineer, decompile, remove or otherwise thwart any protections or disassemble the Licensed Content except and only to the extent that applicable law expressly permits, despite this limitation.

5. **RESERVATION OF RIGHTS AND OWNERSHIP.** Microsoft reserves all rights not expressly granted to you in this agreement. The Licensed Content is protected by copyright and other intellectual property laws and treaties. Microsoft or its suppliers own the title, copyright, and other intellectual property rights in the Licensed Content.

6. **EXPORT RESTRICTIONS.** The Licensed Content is subject to United States export laws and regulations. You must comply with all domestic and international export laws and regulations that apply to the Licensed Content. These laws include restrictions on destinations, end users and end use. For additional information, see www.microsoft.com/exporting.

7. **SUPPORT SERVICES.** Because the Licensed Content is “as is”, we may not provide support services for it.

8. **TERMINATION.** Without prejudice to any other rights, Microsoft may terminate this agreement if you fail to comply with the terms and conditions of this agreement. Upon termination of this agreement for any reason, you will immediately stop all use of and delete and destroy all copies of the Licensed Content in your possession or under your control.

9. **LINKS TO THIRD PARTY SITES.** You may link to third party sites through the use of the Licensed Content. The third party sites are not under the control of Microsoft, and Microsoft is not responsible for the contents of any third party sites, any links contained in third party sites, or any changes or updates to third party sites. Microsoft is not responsible for webcasting or any other form of transmission received from any third party sites. Microsoft is providing these links to third party sites to you only as a convenience, and the inclusion of any link does not imply an endorsement by Microsoft of the third party site.

10. **ENTIRE AGREEMENT.** This agreement, and any additional terms for the Trainer Content, updates and supplements are the entire agreement for the Licensed Content, updates and supplements.

11. **APPLICABLE LAW.**

a. United States. If you acquired the Licensed Content in the United States, Washington state law governs the interpretation of this agreement and applies to claims for breach of it, regardless of conflict of laws principles. The laws of the state where you live govern all other claims, including claims under state consumer protection laws, unfair competition laws, and in tort.

b. Outside the United States. If you acquired the Licensed Content in any other country, the laws of that country apply.

12. LEGAL EFFECT. This agreement describes certain legal rights. You may have other rights under the laws of your country. You may also have rights with respect to the party from whom you acquired the Licensed Content. This agreement does not change your rights under the laws of your country if the laws of your country do not permit it to do so.

13. DISCLAIMER OF WARRANTY. THE LICENSED CONTENT IS LICENSED "AS-IS" AND "AS AVAILABLE." YOU BEAR THE RISK OF USING IT. MICROSOFT AND ITS RESPECTIVE AFFILIATES GIVES NO EXPRESS WARRANTIES, GUARANTEES, OR CONDITIONS. YOU MAY HAVE ADDITIONAL CONSUMER RIGHTS UNDER YOUR LOCAL LAWS WHICH THIS AGREEMENT CANNOT CHANGE. TO THE EXTENT PERMITTED UNDER YOUR LOCAL LAWS, MICROSOFT AND ITS RESPECTIVE AFFILIATES EXCLUDES ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT.

14. LIMITATION ON AND EXCLUSION OF REMEDIES AND DAMAGES. YOU CAN RECOVER FROM MICROSOFT, ITS RESPECTIVE AFFILIATES AND ITS SUPPLIERS ONLY DIRECT DAMAGES UP TO US\$5.00. YOU CANNOT RECOVER ANY OTHER DAMAGES, INCLUDING CONSEQUENTIAL, LOST PROFITS, SPECIAL, INDIRECT OR INCIDENTAL DAMAGES.

This limitation applies to

- o anything related to the Licensed Content, services, content (including code) on third party Internet sites or third-party programs; and
- o claims for breach of contract, breach of warranty, guarantee or condition, strict liability, negligence, or other tort to the extent permitted by applicable law.

It also applies even if Microsoft knew or should have known about the possibility of the damages. The above limitation or exclusion may not apply to you because your country may not allow the exclusion or limitation of incidental, consequential or other damages.

Please note: As this Licensed Content is distributed in Quebec, Canada, some of the clauses in this agreement are provided below in French.

Remarque: Ce le contenu sous licence étant distribué au Québec, Canada, certaines des clauses dans ce contrat sont fournies ci-dessous en français.

EXONÉRATION DE GARANTIE. Le contenu sous licence visé par une licence est offert « tel quel ». Toute utilisation de ce contenu sous licence est à votre seule risque et péril. Microsoft n'accorde aucune autre garantie expresse. Vous pouvez bénéficier de droits additionnels en vertu du droit local sur la protection des consommateurs, que ce contrat ne peut modifier. La ou elles sont permises par le droit locale, les garanties implicites de qualité marchande, d'adéquation à un usage particulier et d'absence de contre-façon sont exclues.

LIMITATION DES DOMMAGES-INTÉRÊTS ET EXCLUSION DE RESPONSABILITÉ POUR LES DOMMAGES. Vous pouvez obtenir de Microsoft et de ses fournisseurs une indemnisation en cas de dommages directs uniquement à hauteur de 5,00 \$ US. Vous ne pouvez prétendre à aucune indemnisation pour les autres dommages, y compris les dommages spéciaux, indirects ou accessoires et pertes de bénéfices.

Cette limitation concerne:

- tout ce qui est relié au le contenu sous licence, aux services ou au contenu (y compris le code) figurant sur des sites Internet tiers ou dans des programmes tiers; et.
- les réclamations au titre de violation de contrat ou de garantie, ou au titre de responsabilité stricte, de négligence ou d'une autre faute dans la limite autorisée par la loi en vigueur.

Elle s'applique également, même si Microsoft connaissait ou devrait connaître l'éventualité d'un tel dommage. Si votre pays n'autorise pas l'exclusion ou la limitation de responsabilité pour les dommages indirects, accessoires ou de quelque nature que ce soit, il se peut que la limitation ou l'exclusion ci-dessus ne s'appliquera pas à votre égard.

EFFET JURIDIQUE. Le présent contrat décrit certains droits juridiques. Vous pourriez avoir d'autres droits prévus par les lois de votre pays. Le présent contrat ne modifie pas les droits que vous confèrent les lois de votre pays si celles-ci ne le permettent pas.

Revised November 2014



Contents

■	Module 0 Welcome	1
	Start Here	1
■	Module 1 Designing a Dependency Management Strategy	7
	Introduction	7
	Packaging Dependencies	11
	Package Management	16
	Implement a Versioning Strategy	26
	Module Review	31
■	Module 2 Manage Security and Compliance	33
	Introduction	33
	Package Security	35
	Open Source Software	38
	Integrating License and Vulnerability Scans	43
	Module Review	46



Module 0 Welcome

Start Here

Azure DevOps Curriculum

Welcome to the **Implementing Dependency Management** course. This course is part of a series of courses to help you prepare for the AZ-400, **Microsoft Azure DevOps Solutions**¹ certification exam.

The DevOps certification exam is for DevOps professionals who combine people, process, and technologies to continuously deliver valuable products and services that meet end user needs and business objectives. DevOps professionals streamline delivery by optimizing practices, improving communications and collaboration, and creating automation. They design and implement strategies for application code and infrastructure that allow for continuous integration, continuous testing, continuous delivery, and continuous monitoring and feedback.

Exam candidates must be proficient with Agile practices. They must be familiar with both Azure administration and Azure development and experts in at least one of these areas. Azure DevOps professionals must be able to design and implement DevOps practices for version control, compliance, infrastructure as code, configuration management, build, release, and testing by using Azure technologies.

AZ-400 Study Areas	Weights
Implement DevOps Development Processes	20-25%
Implement Continuous Integration	10-15%
Implement Continuous Delivery	10-15%
Implement Dependency Management	5 -10%
Implement Application Infrastructure	15-20%
Implement Continuous Feedback	10-15%
Design a DevOps Strategy	20-25%

There are seven exam study areas. Each study area has a corresponding course. While it is not required that you have completed any of the other courses in the DevOps series before taking this course, it is

¹ <https://www.microsoft.com/en-us/learning/exam-AZ-400.aspx>

highly recommended that you start with the first course in the series, and progress through the courses in order.

✓ This course will focus on preparing you for the **Implement Dependency Management** area of the AZ-400 certification exam.

About this Course

Course Description

This course provides the knowledge and skills to implement dependency management. Students will learn how to design a dependency management strategy and manage security and compliance.

Level: Intermediate

Audience

Students in this course are interested in implementing dependency management or in passing the Microsoft Azure DevOps Solutions certification exam.

Prerequisites

- Students should have fundamental knowledge about Azure, version control, Agile software development, and core software development principles. It would be helpful to have experience in an organization that delivers software.
- It is recommended that you have experience working in an IDE, as well as some knowledge of the Azure portal. However, students who may not have a technical background in these technologies, but who are curious about DevOps practices as a culture shift, should be able to follow the procedural and expository explanations of continuous integration regardless.

Expected 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
- Manage security and compliance
- Inspect open source software packages for security and license compliance to align with corporate standards
- Configure build pipeline to access package security and license rating
- Configure secure access to package feeds

Syllabus

This course includes content that will help you prepare for the Microsoft Azure DevOps Solution certification exam. Other content is included to ensure you have a complete picture of DevOps. The course content includes a mix of videos, graphics, reference links, module review questions, and hands-on labs.

Module 1 – Designing a Dependency Management Strategy

- Lesson: Introduction
- Lesson: Packaging Dependencies

- Lesson: Package Management
- Lesson: Implement a Versioning Strategy
- Lab: Updating packages

Module 2 – Manage Security and Compliance

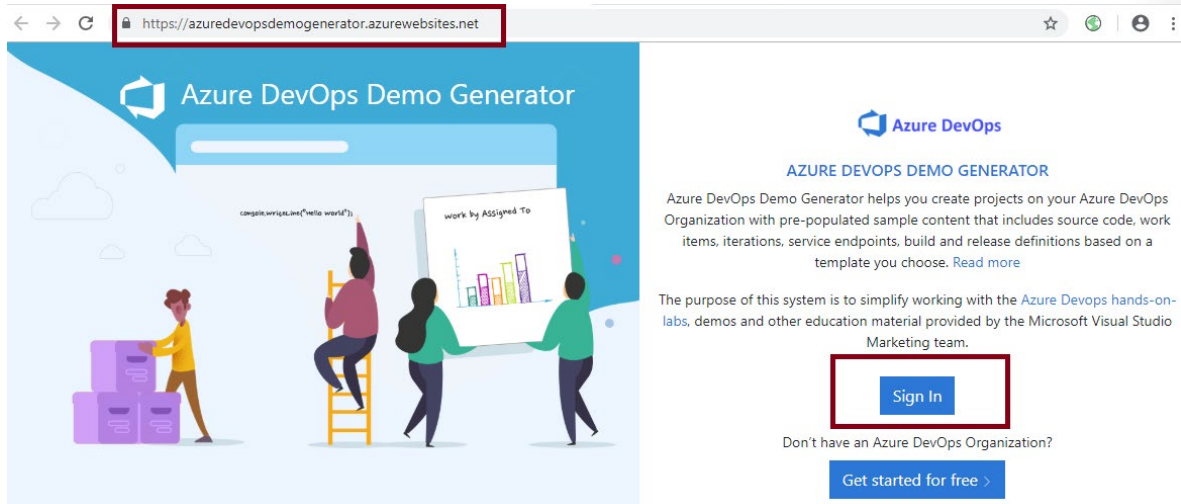
- Lesson: Introduction
- Lesson: Package Security
- Lesson: Open Source Software
- Lesson: Integrating License and Vulnerability Scans

✓ This course uses the **Microsoft DevOps Lab Environment**² to provide a hands-on learning environment.

Lab Environment Setup

We highly recommend that you complete the assigned hands-on lab work. To do the hands-on labs, you will need to complete the following steps.

1. Sign up for a free **Azure DevOps account**³. Use the Sign up for a free account button to create your account. If you already have an account proceed to the next step.
2. Sign up for free **Azure Account**⁴. If you already have an account proceed to the next step.
3. To make it easier to get set up for testing Azure DevOps, a **Azure DevOps Generator Demo**⁵ program has been created. Click the **Sign In** button and sign in with your Azure DevOps account.



4. You will then be asked to confirm that the generator site can have permission to create objects in your Azure DevOps account.

² <https://azuredevopslabs.com/>

³ <https://www.azuredevopslabs.com/>

⁴ <https://azure.microsoft.com/en-us/free/>

⁵ <https://azuredevopsdemogenerator.azurewebsites.net/>

Team dashboards (manage)
Grants the ability to manage team dashboard information

Wiki (read and write)
Grants the ability to read, create and updates wikis, wiki pages and wiki attachments.

[Learn more](#)

If you change your mind at any time, you can manage authorizations on your [profile page](#).

Accept **Deny**

By clicking **Accept**, you allow this app to perform the above actions on your behalf and you agree to Microsoft Terms of Use and Privacy Statement.

5. If you agree, click the **Accept** button and you should be greeted by the Create New Project screen:

Create New Project

Select Organization :

New Project Name :

Selected Template : ...

6. Select the appropriate organization (if you have more than one) and enter **Parts Unlimited** as the **New Project Name**, then click the ellipsis to view the available templates. These will change over time but you should see a screen similar to the following:

Choose a template

General DevOps Labs

SmartHotel360

Scrum aspnetcore azureappservice

This template contains work items, code and pipeline definitions for the public web site of SmartHotel360, an E2E reference sample app with several consumer and line-of-business apps and an Azure backend. For

MyHealthClinic

Scrum aspnetcore azureappservice

This template provisions a scrum based team project with code, work items for a sample ASP.NET Core web application-My Health Clinic. The template also includes pipeline definition to build and deploy the

PartsUnlimited

Scrum aspdotnet azureappservice

Azure SQL

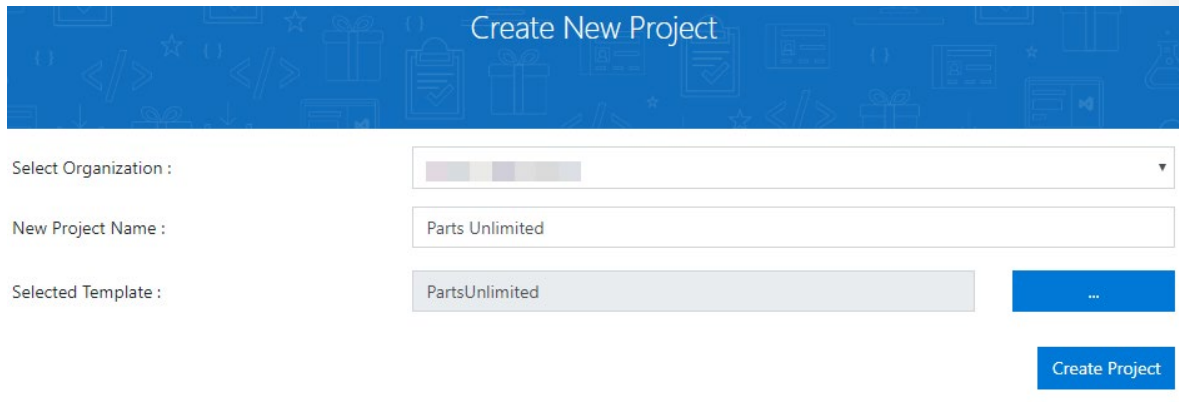
Use this lab to provision a scrum based team project containing sample work items, complete source code and pipeline definitions to develop Parts Unlimited a

MyShuttle

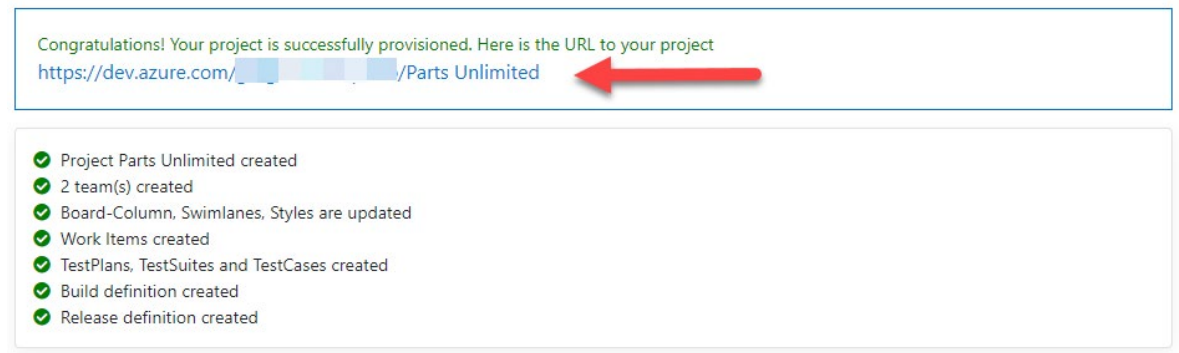
Scrum java application azure web app

MySQL

7. From the **General** tab, choose **PartsUnlimited**, then click **Select Template**.



8. Now that the Create New Project screen is completed, click **Create Project** to begin the project creation phase.



9. When the project is successfully completed, click the link provided to go to your team project within Azure DevOps.
- ✓ Note that because Azure DevOps was previously called VSTS (Visual Studio Team Services), some of the existing hands-on labs might refer to VSTS rather than Azure DevOps.



Module 1 Designing a Dependency Management Strategy

Introduction

Introduction

Hi, and welcome. In this module, we will talk about managing dependencies in software development. We are going to cover what dependencies are and how to identify them in your codebase. Then you will learn how to package these dependencies and manage the packages in package feeds. Finally, you are going to learn about versioning strategies.

Module Objectives

At the end of this module you will have learned:

- 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

We will look at dependency management as a concept in software and why it is needed. We are going to look at dependency management strategies and how you can identify components in your source code and change these to dependencies.

What is dependency management?

Before we can understand dependency management, we will need to first get introduced to the concepts of dependencies.

Dependencies in software

Modern software development involves complex projects and solutions. Projects have dependencies on other projects and solutions are not single pieces of software. The solutions and software built consists of multiple parts and components and are often reused.

As codebases are expanding and evolving it needs to be componentized to be maintainable. A team that is writing software will not write every piece of code by itself, but leverage existing code written by other teams or companies and open source code that is readily available. Each component can have its own maintainers, speed of change and distribution, giving both the creators and consumers of the components autonomy.

A software engineer will need to identify the components that make up parts of the solution and decide whether to write the implementation or include an existing component. The latter approach introduces a dependency on other components.

Why is dependency management needed?

It is essential that the software dependencies that are introduced in a project and solution can be properly declared and resolved. You need to be able to manage the overall composition of project code and the included dependencies. Without proper dependency management it will be hard to keep the components in the solution controlled.

Management of dependencies allows a software engineer and team to be more efficient working with dependencies. With all dependencies being managed it is also possible to stay in control of the dependencies that are consumed, enabling governance and security scanning for use of packages with known vulnerabilities or exploits.

Elements of a dependency management strategy

There are a number of aspects for a dependency management strategy.

- **Standardization**
Managing dependencies benefits from a standardized way of declaring and resolving them in your codebase. Standardization allows a repeatable, predictable process and usage that can be automated as well.
- **Package formats and sources**
The distribution of dependencies can be performed by a packaging method suited for the type of dependency in your solution. Each dependency is packaged using its applicable format and stored in a centralized source. Your dependency management strategy should include the selection of package formats and corresponding sources where to store and retrieve packages.
- **Versioning**
Just like your own code and components, the dependencies in your solution usually evolve over time. While your codebase grows and changes, you need to take into account the changes in your dependencies as well. This requires a versioning mechanism for the dependencies so you can be selective of the particular version of a dependency you want to use.

Identifying dependencies

It starts with identifying the dependencies in your codebase and deciding which dependencies will be formalized.

Your software project and its solution probably already use dependencies. It is very common to use libraries and frameworks that are not written by yourself. Additionally, your existing codebase might have

internal dependencies that are not treated as such. For example, take a piece of code that implements certain business domain model. It might be included as source code in your project, and also consumed in other projects and teams. You need to look into your codebase to identify pieces of code that can be considered dependencies to also treat them as such. This requires changes to how you organize your code and build the solution. It will bring your components.

Source and package componentization

Current development practices already have the notion of componentization. There are two ways of componentization commonly used.

1. Source componentization

The first way of componentization is focussed on source code. It refers to splitting up the source code in the codebase in separate parts and organizing it around the identified components. It works as long as the source code is not shared outside of the project. Once the components need to be shared, it requires distributing the source code or the produced binary artefacts that are created from it.

2. Package componentization

The second way uses packages. Distributing of software components is performed by means of packages as a formal way of wrapping and handling the components. A shift to packages adds characteristics needed for proper dependency management, like tracking and versioning of packages in your solution.

See also¹

Decompose your system

Before you can change your codebase into separate components to prepare for finding dependencies that can be taken out of your system, you will need to get better insights in your code and solution. This allows you to decompose your system to individual components and dependencies.

The goal is to reduce the size of your own codebase and system, making it more efficient to build and manageable in the end. You achieve this by removing certain components of your solution. These are going to be centralized, reused and maintained independently. You will remove those components and externalizing them from your solution at the expense of introducing dependencies on other components.

This process of finding and externalizing components is effectively creating dependencies. It may require some refactoring, such as creating new solution artifacts for code organization, or code changes to cater for the unchanged code to take a dependency on an (external) component. You might need to introduce some code design patterns to isolate and include the componentized code. Examples of patterns are abstraction by interfaces, dependency injection and inversion of control.

Decomposing could also mean that you will replace your own implementation of reusable code with an available open source or commercial component.

Scanning your codebase for dependencies

There are a number of ways to identify the dependencies in your codebase. These include scanning your code for patterns and reuse, as well as analyzing how the solution is composed of individual modules and components.

- **Duplicate code**

When certain pieces of code appear in several places it is a good indication that this code can be reused. Keep in mind that code duplication is not necessarily a bad practice. However, if the code can

¹ <https://docs.microsoft.com/en-us/azure/devops/artifacts/collaborate-with-packages?view=vsts>

be made available in a properly reusable way, it does have benefits over copying code and have to manage that. The first step to isolate these pieces of duplicate code is to centralize them in the codebase and componentize them in the appropriate way for the type of code.

- **High cohesion and low coupling**

A second approach is to find code that might define components in your solution. You will look for code elements that have a high cohesion to each other, and low coupling with other parts of code. This could be a certain object model with business logic, or code that is related because of its responsibility, such as a set of helper or utility code or perhaps a basis for other code to be built upon.

- **Individual lifecycle**

Related to the high cohesion you can look for parts of the code that have a similar lifecycle and can be deployed and released individually. If such code can be maintained by a team separate from the codebase that it is currently in, it is an indication that it could be a component outside of the solution.

- **Stable parts**

Some parts of your codebase might have a slow rate of change. That particular code is stable and is not altered often. You can check your code repository to find the code with a low change frequency.

- **Independent code and components**

Whenever code and components are independent and unrelated to other parts of the system, it can potentially be isolated to a separate component and dependency.

You can use a variety of tools to assist you in scanning and examining your codebase. These range from tools that scan for duplicate code, draw solution dependency graphs, to tools that can compute metrics for coupling and cohesion.

Packaging Dependencies

Package Management

Previously you learned the need for package management and the role that packaging your dependencies plays in your dependency management strategy. Now we will look at more details of working with packages for your dependencies. We will cover package types, feeds, and sources.

Packages

Packages are used to define the components you rely and depend upon in your software solution. They provide a way to store those components in a well defined format with metadata to describe it.

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.

Types of packages

Packages can be used for a variety of components. The type of components you want to use in your codebase differ for the different parts and layers of the solution you are creating. These range from frontend components, such as JavaScript code files, to backend components like .NET assemblies or Java components, complete self-contained solutions or reusable files in general.

Over the past years the packaging formats have changed and evolved. At the moment there are a couple of de facto standard formats for packages.

- **NuGet**

NuGet packages (pronounced “nugget”) is a standard used for .NET code artifacts. This includes .NET assemblies and related files, tooling and sometimes only metadata. NuGet defines the way packages are created, stored and consumed. A NuGet package is essentially a compressed folder structure with files in ZIP format and has the `.nupkg` extension.

[See also: <https://docs.microsoft.com/en-us/nuget/what-is-nuget>]

- **NPM**

An NPM package is used for JavaScript development. It originates from node.js development where it is the default packaging format. A NPM package is a file or folder that contains JavaScript files and a `package.json` file describing the metadata of the package. For node.js the package usually contains one or more modules that can be loaded once the package is consumed.

[See also: <https://docs.npmjs.com/about-packages-and-modules>]

- **Maven**

Maven is used for Java based projects. Each package has It has a Project Object Model file describing the metadata of the project and is the basic unit for defining a package and working with it.

- **Docker**

Docker packages are called images and contain complete and self-contained deployments of components. Most commonly a Docker image represents a software component that can be hosted and

executed by itself, without any dependencies on other images. Docker images are layered and might be dependent on other images as their basis. Such images are referred to as base images.

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`. There are other names in use, such as repository or registry. We will refer to all of these as package feeds, unless it is necessary to use the specific name for clarity.

Each package type has its own type of feed. Put another way, one feed typically contains one type of packages. There are NuGet feeds, NPM feeds, Maven repositories and Docker registries.

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.

Private and public feeds

The package feeds are centralized and available for many different consumers. Depending on the package, its purpose and origin, it might be generally available or to a select audience. Typically open source projects for applications, libraries and frameworks are shared to everyone and publically available. The feeds can be exposed in public or private to distinguish in visibility. Public feeds can be consumed by anyone.

There might be reasons why you do not want your packages to be available publically. This could be because it contains intellectual property or does not make sense to share with other software developers. Components that are developed for internal use might be available only to the project, team or company that developed it.

In such cases you can still use packages for dependency management and choose to store the package in a private package feed.

Private feeds can only be consumed by those who are allowed access.

Package feed managers

Each of the package types has a corresponding manager that takes care of one or more of the following aspects of package management:

- Installation and removal of local packages
- Pushing packages to a remote package feed
- Consuming packages from a remote package feed
- Searching feeds for packages

The package manager have cross-platform command-line interface (CLI) tools to manage the local packages and feeds that host the packages. This CLI tooling is part of a local install on a development machine.

Choosing tools

The command-line nature of the tooling offers the ability to include it in scripts to automate the package management. Ideally, one should be able to use the tooling in build and release pipelines for component creating, publishing and consuming packages from feeds.

Additionally, developer tooling can have integrated support for working with package managers, providing an user interface for the raw tooling. Examples of such tooling are Visual Studio 2017, Visual Studio Code and Eclipse.

Package sources

The various package types have a standard source that is commonly used for public use. It is a go-to place for developers to find and consume publically available components as software dependencies. These sources are package feeds. We will discuss the public and private package sources available to give you a starting point for finding the most relevant feeds.

Public

In general you will find that publically available package sources are free to use. Sometimes they have a licensing or payment model for consuming individual packages or the feed itself.

These public sources can also be used to store packages you have created as part of your project. It does not have to be open source, although it is in most cases. Public and free package sources that offer feeds at no expense will usually require that you make the packages you store publically available as well.

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

Private feeds can be used in cases where packages should be available to a select audience.

The main difference between public and private feeds is the need for authentication. Public feeds can be anonymously accessible and optionally authenticated. Private feeds can be accessed only when authenticated.

There are two options for private feeds:

1. Self-hosting

Some of the package managers are also able to host a feed. Using on-premises or private cloud resources one can host the required solution to offer a private feed.

2. SaaS services

A variety of third party vendors and cloud providers offer software-as-a-service feeds that can be kept privately. This typically requires a consumption fee or cloud subscription.

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

Package type	Self-hosted private feed	SaaS private feed
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

Consume packages

Each software project that consumes packages to include the required dependencies will need to use the package manager and one or more packages sources. The package manager will take care of downloading the individual packages from the sources and install them locally on the development machine or build server.

The developer flow will follow this general pattern:

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.

The package manager tooling will facilitate searching and installing the components in the packages.

How this is performed varies for the different package types. Refer to the documentation of the package manager for instructions on consuming packages from feeds.

To get started you will need to specify the package source to be used. Package managers will have a default source defined that refers to the standard package feed for its type. Alternative feeds will need to be configured to allow consuming the packages they offer.

Upstream sources

Part of the package management involves keeping track of the various sources. It is possible to refer to multiple sources from a single software solution. However, 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. The source will include these downloaded packages and starts to act as a cache for the upstream source. It also offers the ability to keep track of any packages from the external upstream source.

An upstream source can be a way to avoid direct access of developer and build machines to external sources. The private feed uses the upstream source as a proxy to the otherwise external source. It will be

your feed manager and private source that have the communication to the outside. Only privileged roles can add upstream sources to a private feed.

See also²

Packages graph

A feed can have one or more upstream sources, which might be internal or external. Each of these can have additional upstream sources, creating a package graph of source. Such a graph can offer many possibilities for layering and indirection of origins of packages. This might fit well with multiple teams taking care of packages for frameworks and other base libraries.

The downside is that package graphs can become complex when not properly understood or designed. It is important to understand how you can create a proper package graph.

See also³

² <https://docs.microsoft.com/en-us/azure/devops/artifacts/concepts/upstream-sources>

³ <https://docs.microsoft.com/en-us/azure/devops/artifacts/concepts/package-graph>

Package Management

Package management

Previously you learned about packaging dependencies and the various packaging formats, feeds, sources and package managers. Now, you will learn more about package management and how to create a feed and publish packages to it. During this module NuGet and Azure Artifacts are used as an example of a package format and a particular type of package feed and source.

Azure Artifacts

Microsoft Azure DevOps provides various features for application lifecycle management, including work item tracking, source code repositories, build and release pipelines and artifact management.

The artifact management is called `Azure Artifacts` and was previously known as `Package management`. It offers public and private feeds for software packages of various types.

Types of packages supported

Azure Artifacts currently supports feeds that can store 5 different package types:

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

The previous module 1.2 covered the package types for NuGet, NPM, Maven and Python. Universal packages are an Azure Artifacts specific package type. In essence it is a versioned package containing multiple files and folders.

A single Azure Artifacts feed can contain any combination of such packages. You can connect to the feed using the package managers and the corresponding tooling for the package types. For Maven packages this can also be the Gradle build tool.

Selecting package sources

When creating your solution you will decide which packages you want to consume to offer the components you are dependent on. The next step is to determine the sources for these packages. The main choice is selecting public and private feeds or a combination of these.

Publically available packages can usually be found in the public package sources. This would be `nuget.org`, `npmjs` and `pypi.org`. Your solution can select these sources if it only consumes packages available there.

Whenever your solution also has private packages, that are not or cannot be available on public sources, you will need to use a private feed.

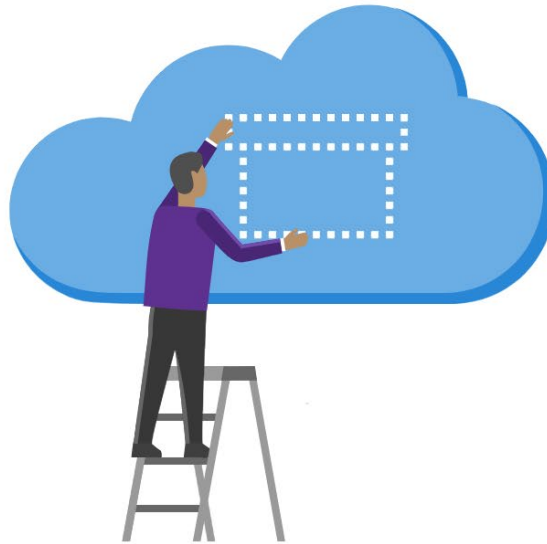
In the previous module you learned that public package sources can be upstream sources to private feeds. Azure Artifacts allows its feeds to specify one or more upstream sources, which can be public or other private feeds.

Publishing packages

As software is developed and components are written, you will most likely also produce components as dependencies that can be packaged for reuse. Discussed previously was guidance to find components that can be isolated into dependencies. These components need to be managed and packaged. After that they can be published to a feed, allowing others to consume the packages and use the components it contains.

Creating a feed

The first step is to create a feed where the packages can be stored. In Azure Artifacts you can create multiple feeds, which are always private. During creation you can specify the name, visibility and whether to prepopulate the default public upstream sources for NuGet, NPM and Python packages.





Create new feed

Feeds host and control permissions for your packages.

Name *

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

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

Controlling access

The Azure Artifacts feed you created is always private and not available publically. You need access to it by authenticating to Azure Artifacts with an account that has access to Azure DevOps and a team project.

By default a feed will be available to all registered users in Azure DevOps. You can select it to be visible only to the team project where the feed is created. Whichever option is chosen, you can change the permissions for a feed from the settings dialog.

Push packages to a feed

Once you have authenticated to Azure DevOps you are allowed to pull and push packages to the package feed, provided you have permissions to do so.

Pushing packages is done with the tooling for the package manager. Each of the package managers and tooling have different syntax for pushing.

To manually push a NuGet package you would use the `NuGet.exe` command-line tool. For a package called `MyDemoPackage` the command would resemble this:

```
nuget.exe push -Source {NuGet package source URL} -ApiKey YourKey YourPackage\YourPackage.nupkg
```

Updating packages

Packages might need to be updated during their lifetime. Technically, updating a package is performed by pushing a new version of the package to the feed. The package feed manager takes care of properly storing the updated package amongst the existing packages in the feed.

Please note that updating packages requires a versioning strategy. This will be covered in the next chapter.

Demonstration Creating a Package Feed

Prerequisite: Open an existing Azure DevOps project

Steps to create a package feed

- Go to dev.azure.com and open your team project
 - Open Azure Artifacts
 - Click on the button **New feed**
 - Enter a name: `PartsUnlimited`
 - Click **Create**

The feed is now created.

- Go to **Settings** and click **Feed Settings**

You are in the tab Feed Details

- Set a description: Packages for the PartsUnlimited Web application
 - Click **Save**
 - Open the Permissions tab

Under permissions you will find which roles are available under which users and groups and you will learn about this in a next module.

- Open the Views tab

We will talk about this later in the module.

- Open the **Upstream sources** tab

You can manage and add your upstream sources here.

- Go back to the root of the feed by clicking on **PartsUnlimited**
 - Click **Connect to Feed**

This dialog shows info about how packages that are stored in this feed can be consumed from various tools. You will learn more about consuming packages in a next demonstration.

Demonstration Pushing a Package

Prerequisite: Make sourcecode for PartsUnlimited available in your Azure DevOps repo

Steps to create a NuGet package

- Go to Visual Studio and open your PartsUnlimited project

Here is a ASP.NET MVC application, where security related parts have been isolated into a seperate .NET standard project. Isolating gives you the option to create a package for this code and publish it to a feed. .NET standard has support for creating Nuget packages.

- Right-click PartsUnlimited.Security project and select **Properties**

Under the tab Package you can set the version and package id.

- Build the project PartsUnlimited.Security
- Go to Command prompt, use command `dir bin\debug`

Here you see the .nupkg file for PartsUnlimited.Security

- Open your team project in dev.azure.com and go to Artifacts

- Click **Connect to feed**
- Follow the dialog instructions
 - Copy the command for "Add this feed" by clicking the **Copy** icon.
- Switch back to your commandline
 - Look at the existing NuGet sources with command: `nuget sources`
You see two NuGet sources available now.
 - Paste and run the copied instructions
 - Look at the existing NuGet sources again with command: `nuget sources`
You see a third NuGet source available.

Steps to publish the package

- Go back to the dialog instructions
 - Copy the command for "Push a Package" by clicking the **Copy** icon.

- Switch back to your commandline and paste the copied instructions
 - Change the folder and name `my_package.nupkg` to `bin\debug\PartsUnlimited.Security1.0.0.nupkg`
 - Run the command

We have published the package to the feed and is pushed succesfully.

- Check if the package is available in Azure DevOps Artifacts
 - Close the dialog instructions
 - Refresh the Artifacts page

You see the succesfully published package.

Demonstration Promoting a Package

Prerequisite: Have acces to an existing Azure DevOps project and the connected package feed from the previous demo

Steps to demonstrate the views that exist on package feeds in Azure Artifacts

- Go to dev.azure.com and open your team project
 - Open **Artifacts** and select the feed **PartsUnlimited.Security**
 - Go to **Settings** and click **Feed Settings**
 - Open the **Views** tab

*By default there will be three views. Local: includes all packaages in the feed and all cached from upstream sources. Prerelease and Release. In the **Default view** column is a check behind Local. This is the default view that will always be used.*

Steps to use the release view instead

- Open Visual Studio and open NuGet Package Manager
 - Click the settings wheel and check the source adress for the PartsUnlimited feed.

If you want to use a different view than the local view, you need to include that in the Source url of your feed, by adding @Release.
 - Add @Release to the source url `../PartsUnlimited@Release/nuget/..` and click **Update**
 - **Refresh** the Browse tab

*You will see there are **No packages found** in the Release feed. Before any packages will appear, you need to promote them.*

Steps to promote packages to particular views

- Go back to your feed in Azure Artifacts
 - Click on the created Nuget Package **PartsUnlimited.Security**
 - Click **Promote**
 - Select the feed you want to use, in this case **PartsUnlimited@Release** and Promote.
 - Go back to the **Overview**

If we look again at our package in the feed, you will notice there is now a Release tag associated with this particular package.

- Go back to Visual Studio, **NuGet Package Manager**
- **Refresh** the Browse tab
- Select **PartsUnlimited.Security** and click **Update** and **OK**

The latest version of this package is now used.

Lab: Updating Packages

To get acquainted with package management using Azure Artifacts, you can do the following hands-on lab. It will let you:

- 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

Hands-on lab⁴

Demonstration Consuming Packages

How to consume packages from an Azure Artifacts package feed?

Prerequisite: Successfully published and pushed a Nuget package to the PartsUnlimited feed from previous Demo

Steps to use the previously created package feed: PartsUnlimited

- Remove the PartUnlimited.Security project from the solution
 - Right-click on project and select **Remove**

⁴ <https://www.azuredevopslabs.com/labs/azuredevops/packagemanagement>

- Go to [Manage Nuget packages]
 - Go to **Settings** and select PartsUnlimited feed
 - Deselect the NuGet.org feed
 - Click **OK**
- Click **Browse** and look for the PartsUnlimited package feed
- Install PartsUnlimited.Security into your project
 - Preview changes: Click **OK**

The package has now been added

- Rebuild the project

It still succeeds, but now uses the package from our Azure Artifacts feed.

Integrating in build pipelines

Why integrate in a build pipeline

Although it is very well possible to manually push packages to a package feed, the process is better performed by a build pipeline. The build pipeline offers full automation for creating the packages and pushing them to the package feed. This includes the following:

- **Quality checks**
The pipeline can perform checks to assess the quality of the packages it produces. By testing the package before it is pushed to a feed it can help in maintaining a high quality of the packages that are pushed.
- **Automate to avoid errors**
The pipeline performs its tasks in a predictable and repeatable fashion, ruling out human errors in this part.
- **Implement a versioning strategy**
In a build pipeline one can formalize the versioning strategy. By using the build tasks and configuring these according to the versioning strategy, each of the package created by pipelines will adhere to the versioning scheme.

Tasks for package types

The Azure Pipelines have a variety of tasks that can help in creating, publishing and consuming packages types. Each package type has its own dedicated tasks that has a specific configuration.

The tasks work with package feeds that might be public or private, internal or external. The feed can be one of the official public sources, or package feeds within Azure Artifacts of either the same team project and organization or an external one.

Packaging in build pipeline

The package management tasks have commands that correspond to the commands of the CLI tooling. They differ slightly, but in general there are two important commands:

- Push
The `push` (or `publish`) command pushes a package to the respective feed.

- Restore

The `restore` (or `install`) command will download a package and restore it locally on the build server. This is a required step before building the source code, as all dependencies need to be restored locally to allow the source code to be built correctly.

Some tasks, such as the NuGet task also allow the creation of a package with the `pack` command, or might have a custom command for use of any of the available CLI commands.

A number of package types require that the tooling be installed first. There are separate tasks to take care of this, or the generic script task can perform the installation.

Package type	Task	Installer
NuGet	Nuget	NuGet Installer
npm	npm	Not required
Maven	Maven	Not required
Python	pip	

Restoring packages in build pipeline

A build pipeline that will get sources for the code that is dependent on packages has a slightly different setup. Like the build pipeline that produces and publishes packages, it is necessary to install the package feed manager tooling on the build machine.

After that, it requires restoring/installing the packages in the codebase. Usually this is specified in a package management document that is part of the source code. For NuGet packages this is in the `packages.json` file for .NET projects, or the project file (`.csproj`, `.vbproj`, `.fsproj`) for .NET Core projects.

You typically need the same package pipeline tasks as for the creating and publishing of the package, but specify the command for `restore` or `install`.



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.

See also⁵

Credentials for build pipeline

When consuming packages from a non-public feed, you may have to specify credentials to authenticate against the feed manager.

For Azure Artifacts feeds and Azure Pipelines, authenticating is handled by Azure DevOps. The integration of the two parts of Azure DevOps makes it fairly transparent.

For private sources outside of Azure DevOps you have to consult the documentation of the specific feed manager and source to learn how to authenticate your identity.

See also⁶

⁵ <https://docs.microsoft.com/en-us/azure/devops/pipelines/packages/nuget-restore>

⁶ <https://docs.microsoft.com/en-us/azure/devops/artifacts/feeds/feed-permissions#package-permissions-in-azure-pipelines>

Implement a Versioning Strategy

Introduction to versioning

Software changes over time. The requirements for the software do not stay the same. The functionality it offers and how it is used will grow, change and adopt based on feedback. The hosting of an application might evolve as well, with new operating systems, new frameworks and versions thereof. The original implementation might contain flaws and bugs.

Whatever reason for change, it is unlikely that software is stable and doesn't need to change. Since the software you build takes dependencies on other components, the same holds true for the components and packages you build or use while building your software.

In order to keep track of which piece of software is currently being used, correct versioning becomes essential to maintaining a codebase.

Versioning is also relevant for dependency management, as it relates to the versions of the packages and the components within. Each dependency is clearly identified by its name and version. It allows keeping track of the exact packages being used. Each of the packages has its own lifecycle and rate of change.

Immutable packages

As packages get new versions, your codebase can choose when to use a new version of the packages it consumes. It does so by specifying the specific version of the package it requires. This implies that packages themselves should always have a new version when they change.

Whenever a package is published to a feed it should not be allowed to change any more. If it were, it would be at the risk of introducing potential breaking changes to the code. In essence, a published package is considered to be immutable. Replacing or updating an existing version of a package is not allowed.

Most of the package feeds do not allow operations that would change an existing version. Regardless of the size of the change a package can only be updated by the introduction of a new version. The new version should indicate the type of change and impact it might have.

See also⁷

Versioning of artifacts

It is proper software development practices to indicate changes to code with the introduction of an increased version number. However small or large a change, it requires a new version. A component and its package can have independent versions and versioning schemes.

The versioning scheme can differ per package type. Typically, it uses a scheme that can indicate the type of change that is made. Most commonly this involves three types of changes:

- **Major change**
Major indicates that the package and its contents have changed significantly. It often occurs at the introduction of a complete new version of the package. This can be at a redesign of the component. Major changes are not guaranteed to be compatible and usually have breaking changes from older versions. Major changes might require a substantial amount of work to adopt the consuming codebase to the new version.
- **Minor change**
Minor indicates that the package and its contents have substantial changes made, but are a smaller

⁷ <https://docs.microsoft.com/en-us/azure/devops/artifacts/artifacts-key-concepts#immutability>

increment than a major change. These changes can be backward compatible from the previous version, although they are not guaranteed to be.

- **Patch**

A patch or revision is used to indicate that a flaw, bug or malfunctioning part of the component has been fixed. Normally, this is a backward compatible version compared to the previous version.

How artifacts are versioned technically varies per package type. Each type has its own way of indicating the version in metadata. The corresponding package manager is able to inspect the version information. The tooling can query the package feed for packages and the available versions.

Additionally, a package type might have its own conventions for versioning as well as a particular versioning scheme.

See also⁸

Semantic versioning

One of the predominant ways of versioning is the use of semantic versioning. It is not a standard per se, but does offer a consistent way of expressing intent and semantics of a certain version. It describes a version in terms of its backward compatibility to previous versions.

Semantic versioning uses a three part version number and an additional label. The version has the form of `Major.Minor.Patch`, corresponding to the three types of changes covered in the previous section. Examples of versions using the semantic versioning scheme are `1.0.0` and `3.7.129`. These versions do not have any labels.

For prerelease versions it is customary to use a label after the regular version number. A label is a textual suffix separated by a hyphen from the rest of the version number. The label itself can be any text describing the nature of the prerelease. Examples of these are `rc1`, `beta27` and `alpha`, forming version numbers like `1.0.0-rc1` as a prerelease for the upcoming `1.0.0` version.

Prereleases are a common way to prepare for the release of the label-less version of the package. Early adopters can take a dependency on a prerelease version to build using the new package. In general it is not a good idea to use prerelease version of packages and their components for released software.

It is good to anticipate on the impact of the new components by creating a separate branch in the codebase and use the prerelease version of the package. Changes are that there will be incompatible changes from a prerelease to the final version.

See also⁹

See also¹⁰

Release views

When building packages from a pipeline, the package needs to have a version before the package can be consumed and tested. Only after testing is the quality of the package known. Since package versions cannot and should not be changed, it becomes challenging to choose a certain version beforehand.

Azure Artifacts recognizes a quality level of packages in its feeds and the difference between prerelease and release versions. It offers different views on the list of packages and their versions, separating these based on their quality level. It fits well with the use of semantic versioning of the packages for predictabil-

⁸ <https://docs.microsoft.com/en-us/azure/devops/pipelines/artifacts/nuget#package-versioning>

⁹ <https://docs.microsoft.com/en-us/azure/devops/pipelines/artifacts/nuget#package-versioning>

¹⁰ <https://semver.org/>

ity of the intent of a particular version, but is additional metadata from the Azure Artifacts feed called a `descriptor`.

Feeds in Azure Artifacts have three different views by default. These view are added at the moment a new feed is created. The three views are:

- **Release**
The `@Release` view contains all packages that are considered official releases.
- **Prerelease**
The `@Prerelease` view contains all packages that have a label in their version number.
- **Local**
The `@Local` view contains all release and prerelease packages as well as the packages downloaded from upstream sources.

Using views

You can use views to offer help consumers of a package feed to filter between released and unreleased versions of packages. Essentially, it allows a consumer to make a conscious decision to choose from released packages, or opt-in to prereleases of a certain quality level.

By default the `@Local` view is used to offer the list of available packages. The format for this URI is:

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

When consuming a package feed by its URI endpoint, the address can have the requested view included. For a specific view, the URI includes the name of the view, which changes to be:

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

The tooling will show and use the packages from the specified view automatically.

Tooling may offer an option to select prerelease versions, such as shown in this Visual Studio 2017 NuGet dialog. This does not relate or refer to the `@Prerelease` view of a feed. Instead, it relies on the presence of prerelease labels of semantic versioning to include or exclude packages in the search results.

See also¹¹

See also¹²

Promoting packages

Azure Artifacts has the notion of promoting packages to views as a means to indicate that a version is of a certain quality level. By selectively promoting packages you can plan when packages have a certain quality and are ready to be released and supported by the consumers.

You can promote packages to one of the available views as the quality indicator. The two views `Release` and `Prerelease` might be sufficient, but you can create more views when you want finer grained quality levels if necessary, such as `alpha` and `beta`.

¹¹ <https://docs.microsoft.com/en-us/azure/devops/artifacts/concepts/views>

¹² <https://docs.microsoft.com/en-us/azure/devops/artifacts/feeds/views>

Packages will always show in the Local view, but only in a particular view after being promoted to it. Depending on the URL used to connect to the feed, the available packages will be listed.

Upstream sources will only be evaluated when using the @Local view of the feed. Views After they have been downloaded and cached in the @Local view, you can see and resolve the packages in other views after they have promoted to it.

It is up to you to decide how and when to promote packages to a specific view. This process can be automated by using a Azure Pipelines task as part of the build pipeline.

Packages that have been promoted to a view will not be deleted based on the retention policies.

Demonstration Pushing From the Pipeline

Pushing NuGet packages from the Azure DevOps pipeline

Prerequisite: In your Azure DevOps PartsUnlimited project, prepare two builds pipelines (used in previous demos)

1. Build pipeline **PartsUnlimited Security Package**

- **.NET Core** build task (Restore, build, push)
- enable CI trigger

2. Build pipeline **PartsUnlimited E2E**

- **ASP.net** web application type
- NuGet restore task

Steps to build and push the Nuget package

• Edit the build pipeline **PartsUnlimited Security Package**

- dotnet restore
- dotnet build
- dotnet push
- Use the command `nuget push` and specify path `**/*.nupkg`
- Select target feed PartsUnlimited

• Start a new build and select agent pool

*The build has succeeded, but you will see the final step **dotnet push** fails. The reason for this failure can be found in the log information.*

• Open the log information

*It shows the feed already contains the **PartsUnlimited.Security 1.0.0**. We go back to the Visual Studio project to see what is happening.*

• Open the source code for the PartsUnlimited package in Visual Studio in a separate solution

- Open the **Project Properties**
- Go to the package tab

Look at Package version, we see that the version is still 1.0.0. Packages are immutable. As soon as a package is published to a feed there can never be another package with the exact same version. We need to upgrade the version to a new one that uses the major, minor and the changed patch version.

- Change the patch version: 1.0.1

Make a small edit to the code for illustrative purposes, and check in the new code.

- Change the **exception type** check in, commit and push the new code

We go back to the Azure DevOps pipeline. Since there is a trigger on the code we just changed, the build will automatically start.

- Go back to build pipeline for **PartsUnlimited Security Package**

As you see the build is triggered and completed successfully. Including the push to the NuGet feed. Since there was no version conflict, we were able to successfully push the new version to the feed.

- Open **Artifacts** and show the feed

Since there was a successful build for the entire web application, you can see that the PartsUnlimited feed now also includes all the downloaded upstream packages from the NuGet.org source.

- Scroll down and click on the **PartsUnlimited.Security 1.0.1** package

By clicking on it we can inspect the details and the versions.

- Edit the build pipeline **PartsUnlimited E2E**

- Click **NuGet restore**

*There is a second pipeline that builds the complete web application. It is an ASP.net web application build. The NuGet restore task is configured to also consume packages from the PartsUnlimited feed. Because PartsUnlimited has an upstream source for NuGet.org we do not have to **Use packaged from NuGet.org** explicitly. You can uncheck this box.*

Best practices for versioning

There are a number of best practices to effectively use versions, views and promotion flow for your versioning strategy. Here are a couple of suggestions:

- Have a documented versioning strategy
- Adopt SemVer 2.0 for your versioning scheme
- Each repository should have a unique feed
- When creating a package, also publish it to the unique feed

It is good to adopt a best practices yourself and share these in your development teams. It can be made part of the Definition of Done for work items that relate to publishing and consuming packages from feeds.

See also¹³

¹³ <https://docs.microsoft.com/en-us/azure/devops/artifacts/concepts/best-practices>

Module Review

Knowledge Checks

Note that these knowledge check questions are for you to check your own knowledge. They are not part of the grading for this course.

1. If you are creating a feed that will allow yourself and those that you invite to publish, what visibility should you choose?

Suggested Answer

Private

2. Can you create a package feed for Maven in Azure Artifacts?

Suggested Answer

Yes

3. What type of package should you use for Machine learning training data & models?

Suggested Answer

Universal

4. If an existing package is found to be broken or buggy, how should it be fixed?

Suggested Answer

Publish a new version

5. What is meant by saying that a package should be immutable?

Suggested Answer

A published version should never be changed, only replaced by a later version



Module 2 Manage Security and Compliance

Introduction

Introduction

Hello, and welcome to this module about managing security and compliance. In this module, we will talk about how you can secure your packages and feeds and check security requirements on the packages used in developing your software solutions. Also we will cover how to make sure the packages used are compliant to the standard and requirements that exist in your organization from a licensing and security vulnerability perspective.

Learning objectives

At the end of this module, students will be able to:

- Secure access to your packages and feeds
- Explain the implications of using open source software
- Explain open source package licensing
- Integrate license and vulnerability scans
- Work with artifact repositories

Introduction to package security and compliance

In the previous module you learned about hosting package management solutions and working with package feeds and sources. Now we will revisit the hosting and consumption of package feeds from a security and compliance perspective.

Package security

Package feeds are a trusted source of packages. The packages that are offered will be consumed by other code bases and used to build software that needs to be secure. Imagine what would happen if a package feed would offer malicious components in its packages. Each consumer would be affected when installing the packages onto its development machine or build server. This also happens at any other device that will run the end product, as the malicious components will be executed as part of the code. Usually the code runs with high privileges, giving a substantial security risk if any of the packages cannot be trusted and might contain unsafe code.

Therefore, it is essential that package feeds are secured for access by authorized accounts, so only verified and trusted packages are stored there. No one should be able to push packages to a feed without the proper role and permissions. This prevents others from pushing malicious packages. It still assumes that the persons who are allowed to push packages will only add safe and secure packages. Especially in the open source world this is performed by the community. A package source can further guard its feed with the use of security and vulnerability scan tooling. Additionally, consumers of packages can use similar tooling to perform the scans themselves.

Another aspect of security for package feeds is about public or private availability of the packages. The feeds of public sources are usually available for anonymous consumption. Private feeds on the other hand have restricted access most of the time. This applies to consumption and publishing of packages. Private feeds will allow only users in specific roles or teams access to its packages.

In the next chapter we will look at package security and how to secure package feeds.

Package compliance

Nowadays companies have obligations towards their customers and employees to make sure that the services they offer with software and IT are compliant to rules and regulations. In part the rules and regulations come from governments, certification and standards institutes. They might also be self imposed rules and regulations from the company or organization itself. This could include rules about how open source is used, which license types are allowed and a package version policy.

When development teams are empowered to make their own choices for the use of packages it becomes very important that they are choosing the right type of packages. In this case that would imply that the packages are allowed from a licensing perspective and follow the chosen ruleset to be compliant with the applicable policies. It involves practices, guidelines, hosting, additional work and introduction of tooling that will help to make sure that compliancy is part of the software development process when it comes to producing and consuming packages.

The compliancy of packages should be guaranteed and provable. The software development processes should take this into account in an integral fashion. We will look at open source software and licensing in the next chapters, and see how we can leverage Azure DevOps and other tools and services to implement a policy for security and compliancy for closed and open source alike.

Package Security

Package security

Securing access to package feeds

Package feeds need to have secure access for a variety of reasons. The access should involve allowing:

- **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

Azure Artifacts has four different roles for package feeds. These are incremental in the permissions they give.

The roles are in incremental order:

- **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

When creating an Azure Artifacts feed, the `Project Collection Build Service` is given contributor rights by default. This organization-wide build identity in Azure Pipelines is able to access the feeds it needs when running tasks. If you changed the build identity to be at project level, you will need also give that identity permissions to access the feed.

Any contributors to the team project are also contributors to the feed. Project Collection Administrators and administrators of the team project, plus the creator of the feed are automatically made owners of the feed. The roles for these users and groups can be changed or removed.

Permissions

The feeds in Azure Artifacts require permission to the various features it offers. The list of permissions consists of increasing privileged operations.

The list of privileges is as follows:

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				✓



For each permission you can assign users, teams and groups to a specific role, giving the permissions corresponding to that role. You need to have the Owner role to be able to do so. Once an account has access to the feed from the permission to list and restore packages it is considered a `Feed user`.

DevOpsCertificationFeed > Feed settings

Feed details	Permissions	Views	Upstream sources	+ Add users/groups	Delete	...
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				

Just like permissions and roles for the feed itself, there are additional permissions for access to the individual views. Any feed user has access to all the views, whether the default views of `@Local`, `@Release` or `@Prerelease`, or newly created ones. During creation of a feed you can choose whether the feed is visible to people in your Azure DevOps organization or only specific people.

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

See also¹

Credentials

Azure DevOps users will authenticate against Azure Active Directory when accessing the Azure DevOps portal. After being successfully authenticated, they will not have to provide any credentials to Azure Artifacts itself. The roles for the user, based on its identity, or team and group membership, are for authorization. When access is allowed, the user can simply navigate to the Azure Artifacts section of the team project.

The authentication from Azure Pipelines to Azure Artifacts feeds is taken care of transparently. It will be based upon the roles and its permissions for the build identity. The previous section on `Roles` covered some details on the required roles for the build identity.

The authentication from inside Azure DevOps does not need any credentials for accessing feeds by itself. However, when accessing secured feeds outside Azure Artifacts, such as other package sources, you most likely have to provide credentials to authenticate to the feed manager. Each package type has its own way of handling the credentials and providing access upon authentication. The command-line tooling will provide support in the authentication process.

For the build tasks in Azure Pipelines, you will provide the credentials via a Service connection.

¹ <https://docs.microsoft.com/en-us/azure/devops/artifacts/feeds/feed-permissions>

Open Source Software

Open source software

Let's look at using open-source software in building software.

Using open-source software

Packages contain components that are built from source code. Open source code is publicly available for inspection, reuse and contribution. Most commonly open source projects indicate how the sources can be used and distributed afterwards. A license agreement accompanies the source code and specifies what can and cannot be done.

How software is built

In the previous module we covered how modern software is built by using components. These components are created in part by the team that is writing the entire software solution. Some dependencies are on components created and made available by other teams, third party companies and the community. The packages that contain the components are a formalized way for distribution.

On average the software solution being built is around 80% based on components that exist and are maintained outside of the project. The rest of the solution consists of your code with business logic and specifics for the functional requirements, plus "glue" code that binds together the components and your code.

The components can be a commercial offering or free of charge. A considerable part of the publically available and free components are community efforts to offer reusable components for everyone to use and build software. The persons creating and maintaining these components often also make the source code available. This is considered to be open-source code as opposed to closed source. Closed source means that the source code is not available, even though components are available.

What is open-source software?

Wikipedia defines open-source software as follows:

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.

The related open source software development is a collaborative form of software development, involving multiple contributors. Together they create and maintain software and source code using open sources. The use of open-source software is widely adopted now.

See also²

Microsoft itself has also embraced open-source software in its own software and the development platforms they offer. The .NET platforms, such as the original .NET Framework and even more so .NET Core, use several components that are created by the open source community and not Microsoft itself. In ASP.NET and ASP.NET Core, many of the frontend development libraries are open-source components, such as jQuery, Angular and React. Instead of creating new components themselves, the teams at Microsoft are leveraging the open-source components and take a dependency on them. The teams are also

² https://en.wikipedia.org/wiki/Open-source_software

contributing and investing to the open source components and projects, joining in on the collaborative effort.

Besides adopting external open-source software Microsoft has also made significant parts of their software available as open-source. .NET is a perfect example of how Microsoft has changed its posture towards open-source. It has made the codebase for the .NET Framework and .NET Core available, as well as many other components. The .NET Foundation aims to advocate the needs, evangelize the benefits of the .NET platform and promote the use of .NET open source for developers.

See also³

Challenge to corporates

All in all, modern software development, including the Microsoft developer platform and ecosystem implies the use of open-source components. This has implications for companies that build software, either commercially or for internal use. The inclusion of software components that are not build by the companies themselves, means that there is no full control over the sources.

Others being responsible for the source code that is used in components used within a company, means that you have to accept the risks involved with it. The source code could:

- **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.

The companies will have to make a trade-off: its developers want to be able to use open-source software components, allowing them to speed up development and use modern frameworks, libraries and practices. On the other hand, giving the developers and projects the freedom to include open-source software should not put the company at risk. The challenges to the company are in finding a way to keep the developers empowered and free to choose technology to use, while making sure the risks for the company are managed as well as possible.

Other challenges comes from companies that offer open-source software to the public. These challenges include having a business model around the open-source, when to publish open-source code and how to deal with community contributions. The fact that your source code is open, doesn't imply that anyone can make changes to it. There can be contributions from community collaboration, but a company does not necessarily have to accept it. This is referred to as closed open-source. Suggestions for change are welcome, but the maintainers are the one that carry out the actual changes.

³ <http://www.dotnetfoundation.org>

Licenses and vulnerabilities

Open-source software and the related source code is accompanied by a license agreement. The license describes the way the source code and the components built from it can be used and how any software created with it should deal with it.

According to the open source definition of OpenSource.org a license should not:

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

and some other aspects.

See also⁴

To cover the exact terms of a license several types exist. Each type has its own specifics and implications, which we will cover in the next part.

Even though open-source software is generally developed by multiple contributors from the community, it does not guarantee that the software is secure and without vulnerabilities. Chances are they are discovered by being inspected by multiple reviewers, but the discovery might not be immediate or before being consumed by others.

Since the source code is open-source, people with malicious intent can also inspect the code for vulnerabilities and exploit these when possible. In that regard, it is both a blessing and a curse that open-source software has source code available for others.

Open-source package licensing

In the current and previous module we have talked about software components from the perspective of packages. Packages are the formalized ways to distribute software components. The licensing types and concerns about vulnerabilities extend to the packages, as these contain the components.

Types of licenses

There are multiple licenses used in open-source and they are different in nature. The license spectrum is a chart that shows licenses from the perspective of the developer and the implications of use for downstream requirements that are imposed on the overall solution and source code.

⁴ <http://opensource.org/osd>



On the left side there are the “attribution” licenses. They are permissive in nature and allow practically every type of use by the software that consumes it. An example is building commercially available software including the components or source code under this license. The only restriction is that the original attribution to the authors remains included in the source code or as part of the downstream use of the new software.

The right side of the spectrum shows the “copyleft” licenses. These licenses are considered viral in nature, as the use of the source code and its components, and distribution of the complete software, implies that all source code using it should follow the same license form. The viral nature is that the use of the software covered under this license type forces you to forward the same license for all work with or on the original software.

The middle of the spectrum shows the “downstream” or “weak copyleft” licenses. It also requires that when the covered code is distributed, it must do so under the same license terms. Unlike the copyleft licenses this does not extend to improvements or additions to the covered code.

License implications

Any form of software that uses code or components must take into account the license type that is covered. For companies it becomes important to know the types of licenses for components and packages that developers choose to use. When these include even one viral license, it requires that all software must be using the same license. Any intellectual property you might have, must be made public and open-source according to the terms of the viral license type. This has tremendous impact on the source code of the project and consequently for the company that produces it.

License rating

Licenses can be rated by the impact that they have. When a package has a certain type of license, the use of the package implies keeping to the requirements of the package. The impact the license has on the downstream use of the code, components and packages can be rated as High, Medium and Low, depending on the copy-left, downstream or attribution nature of the license type.

For compliancy reasons, a high license rating can be considered a risk for compliancy, intellectual property and exclusive rights.

Package security

The use of components creates a software supply chain. The resultant product is a composition of all its parts and components. This applies to the security level of the solution as well. Therefore, similar to license types it is important to know how secure the components being used are. If one of the components used is not secure, then the entire solution isn't either. We will talk more on package security and vulnerabilities in the next chapter.

Integrating License and Vulnerability Scans

Artifact repositories

Let's learn about artifact repositories and their use, plus tooling to help in identifying security vulnerabilities and license and compliancy issues.

Looking at package management and the challenges of keeping control of licenses and vulnerabilities, every feed and its packages that is consumed is a potential risk. Normally, the build tooling will interact with internal and potentially external package feeds. Upstream sources can be a way to avoid direct access to external feeds.

If each developer, team and build process would be able to connect directly to external feeds, it is hard to perform package management from the perspective of licensing and security vulnerabilities.

Artifact repositories are used to store software artifacts in a centralized, internal place or flow, much like package feeds. The repository becomes the single source of packages for the entire organization, facilitating an overview of the packages in use by each of the consumers.

Tools for assessing package security and license rating

There are several tools available from third parties to help in assessing the security and license rating of software packages that are in use.

One approach by these tools is to provide the centralized artifact repository as discussed in the previous section. Scanning can be done at any particular time, inspecting the packages that are part of the repository.

The second approach is to use tooling that scans the packages as they are used in a build pipeline. During the build process, the tool can scan the packages by the particular build, giving instantaneous feedback on the packages in use.

Inspect packages in delivery pipeline

There is tooling available to perform security scans on packages, components and source code while running a delivery pipeline. Often such tooling will use the build artifacts during the build process and perform scans.

The tooling can take either of the approaches of working on a local artifact repository or the intermediary build output. Some examples for each are products like:

Tool	Type
Artifactory	Artifact repository
SonarCube	Artifact repository
WhiteSource (Bolt)	Build scanning

Configure pipeline

The configuration of the scanning for license types and security vulnerability in the pipeline is done by using appropriate build tasks in your DevOps tooling. For Azure DevOps these are build pipeline tasks.

WhiteSource is a third party product that offers both a paid and free version to use in Azure Pipelines. The tool uses the local build artifacts on the build server and runs directly from there. It will scan for the various package types used in the build and analyze those found. This requires external connectivity. The results of the analysis are returned in the build results as part of the step for the task.

Demonstration Whitesource Bolt

How to integrate scanning for security vulnerabilities and license ratings using WhiteSource Bolt

Prerequisite: In your Azure DevOps PartsUnlimited project, prepare the build (used in previous demos)

1. Build pipeline **PartsUnlimited E2E**

- **ASP.net** web application
- NuGet restore task configured

Steps to integrate scanning for security vulnerabilities and license ratings

• Edit the build pipeline **PartsUnlimited E2E**

- Add task
- Search for `whitesource` select **WhiteSource Bolt** and add to pipeline
- Place task after Test Assemblies

We can scan for security vulnerabilities and license ratings after it has been successfully tested.

- Save & queue this build

WhiteSource Bolt will now automatically scan for us during a build.

• Open the build and inspect the results from **WhiteSource Bolt**

The build has completed successfully, we can inspect the results from WhiteSource Bolt. It was able to scan all dependencies and upload it for inspection. It also builds a report.

- Open the tab **WhiteSource Bolt Build Report**

This indicates there were some vulnerabilities. Show the vulnerabilities. Look at the license risk and compliance perspective column and show risk levels. These medium rated risks are for further inspection. Take a look at outdated libraries, those are mostly NuGet packages that can be upgraded.

_Using WhiteSource Bolt you can verify if there are any security vulnerabilities or license issues in the code you use or the packages you consume.

Interpret Alerts from Assessment

Interpret alerts from assessment

To correctly interpret the results of scanning tools, you need to be aware of some aspects:

• **False positives**

It is important to verify the findings to be real positives in terms of the scan results. The tooling is an automated way to scan and might be misinterpreting certain vulnerabilities. In the triaging of the finding in the scan results, you should be aware that some findings might not be correct. Such results are called `false positives`, established by human interpretation and expertise. One must not declare a result a false positive too easily. On the other hand, scan results are not guaranteed to be 100% accurate.

- **Security bug bar**

Most likely there will be many security vulnerabilities detected. Some of these *false positives*, but still a lot of findings. Quite often there are more findings than can be handled or mitigated, given a certain amount of time and money. In such cases it is important that there is a security bug bar, indicating the level of vulnerabilities that must be fixed before the security risks are acceptable enough to take the software into production. The bug bar makes sure that it is clear what must be taken care of, and what might be done if there is time and resources left.

The WhiteSource Bolt tooling gives aggregated reporting that can be found under the Pipelines section. It shows three different sections:

- **Security vulnerabilities**

The results from the scan are usually distinguishing between various levels of severity. The severity is a good candidate for establishing the bug bar. For example, the security bug bar could be set to not allow any findings on High or Medium vulnerability level before releasing software to production.

- **License risks and compliancy**

A list of license types that are used. The higher the risk level of the licenses in use that are identified, the bigger chance of compliancy issues.

- **Outdated libraries**

Libraries that are not using the latest version. Although these do not pose a security risk (yet), they might cause maintenance problems and introduce other risks for the software that uses these libraries, components and packages they come in.

The results of the scan tooling will be the basis for selecting what work remains to be done before software is considered stable and done. By setting a security bug bar in the Definition of Done and specifying the allowed license ratings, one can use the reports from the scans to find the work for the development team.

Module Review

Knowledge Checks

Note that these knowledge check questions are for you to check your own knowledge. They are not part of the grading for this course.

1. What issues are often associated with the use of open source libraries?

Suggested Answer

Bugs, security vulnerabilities, and licensing issues

2. How can an open source library cause licensing issues if it is free to download?

Suggested Answer

Each library has usage restrictions as part of the licensing. These restrictions might not be compatible with your intended application use.

3. What is the minimum feed permission that will allow you to list available packages and to install them?

Suggested Answer

Reader

4. What is open source software?

Suggested Answer

A type of software where users of code are permitted to study, change, and distribute the software. The open source license type can limit the actions (such as sale provisions) that can be taken.

5. How can you restrict which files are uploaded in a universal package feed?

Suggested Answer

By using an .artifactignore file