DevOps Kata

**Continuous configuration / package management**

Last updated: 12/12/2016



1. **TABLE OF CONTENT**

[Overview 3](#_Toc469294850)

[Prerequisites 3](#_Toc469294852)

[Exercises 3](#_Toc469294853)

[Exercise 1: Azure Automation 4](#_Toc469294854)

[Task 1: create an azure automation account 4](#_Toc469294855)

[Task 2: Test azure automation account 5](#_Toc469294856)

[Task 3: Create a webhook and verify in VSTS 5](#_Toc469294857)

[Exercise 2: Use and create packages 9](#_Toc469294858)

[Note: You must have installed the package management extension from the VSTS markeplace to get this started. 9](#_Toc469294859)

[Task 1: Create a feed 9](#_Toc469294860)

[Task 2: Publish a package 10](#_Toc469294861)

[Task 3: Add the feed to your NuGet configuration in Visual Studio 12](#_Toc469294862)

[Task 4: Consume your feed in Visual Studio 13](#_Toc469294863)

## Overview

### Microsoft Azure Automation provides a way for users to automate the manual, long-running, error-prone, and frequently repeated tasks that are commonly performed in a cloud and enterprise environment. It saves time and increases the reliability of regular administrative tasks and even schedules them to be automatically performed at regular intervals. You can automate processes using runbooks or automate configuration management using Desired State Configuration.

Componentization is the act of separating and structuring of your product into a set of components: discrete parts of your codebase that provide a set of features. Most .NET projects already have some notion of components in the form of the projects in your solution. For example, a simple website might have a front-end component, a data access component, and a model/data storage component. At a high level, we’ve found the most success when source composition is used for related projects that are worked on by a single team (or a group of related teams) and binary composition is used for OSS, externals (components from faraway or isolated teams), and isolated shared components.

### Prerequisites

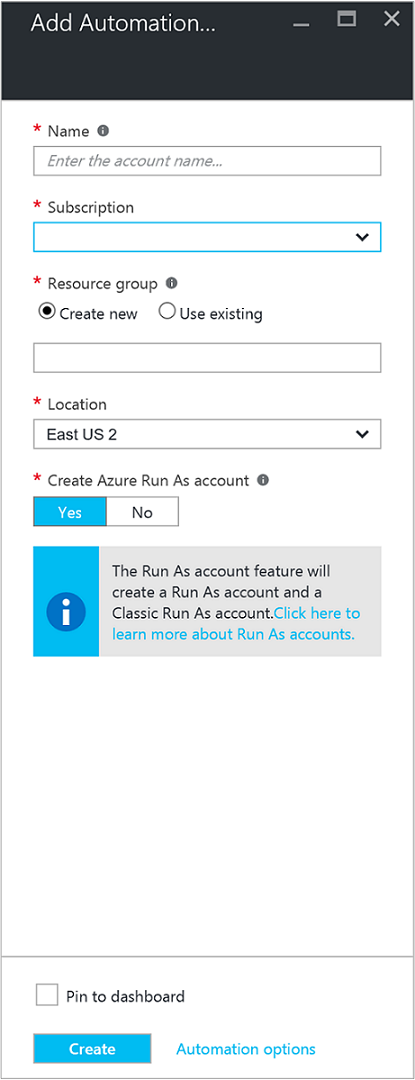
1. In order to complete the lab
2. 1. Log on to your Visual Studio / MSDN subscription and create or use your own VSTS instance. Alternatively, you can use a team sandbox VSTS instance if you have one.
3. 2. Have an azure subscription available that you can deploy resource to.
4. 3. Visual Studio Enterprise
5. 4. Install package management for your VSTS instance.

### Exercises

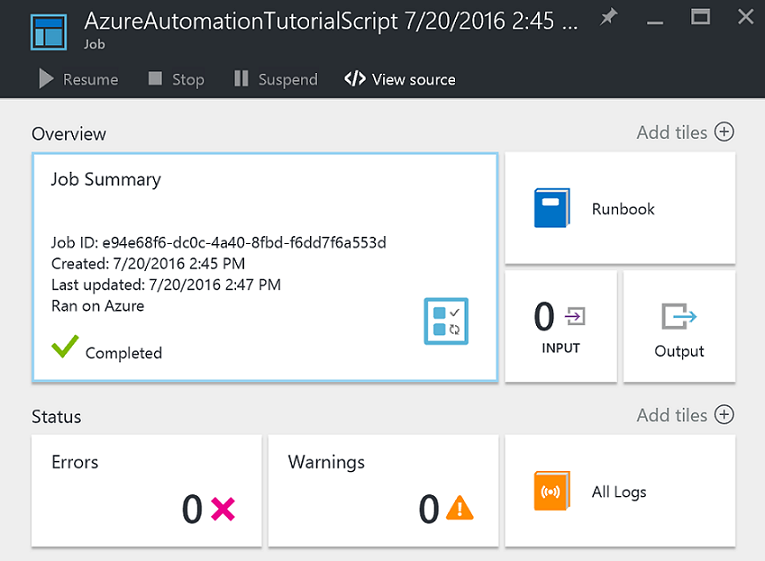
* 1. This hands-on lab includes the following exercises:
  2. Use Azure automation
  3. Use and create packages
  4. Estimated time to complete this exersize: **15 minutes**.

Exercise 1: Azure Automation

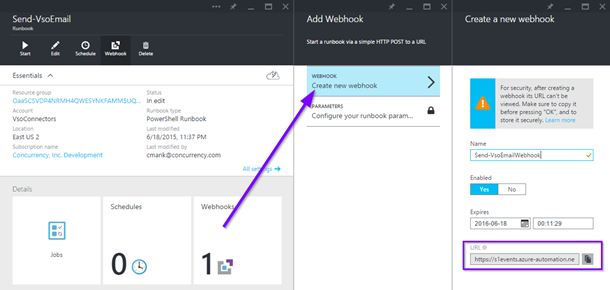
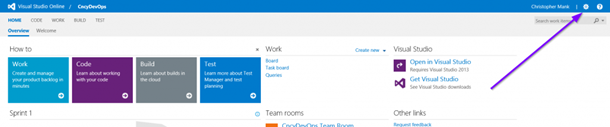
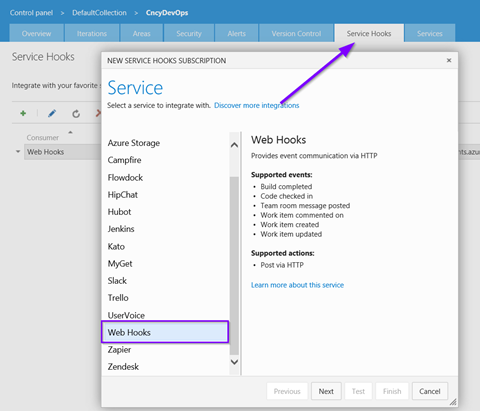
#### Task 1: create an azure automation account

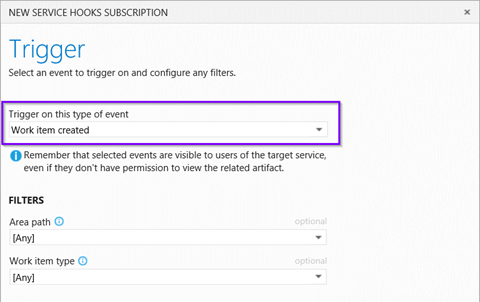
1. In this section, you will perform the following steps to create a new Azure Automation account from the Azure portal. This creates both the Run As and classic Run As account. +
2. Log in to the Azure portal with an account that is a member of the Subscription Admins role and co-administrator of the subscription.
3. Select **Automation Accounts**.
4. In the Automation Accounts blade, click **Add**.  
   
5. In the **Add Automation Account** blade, in the **Name** box type in a name for your new Automation account.
6. If you have more than one subscription, specify one for the new account, as well as a new or existing **Resource group** and an Azure datacenter **Location**.
7. Verify the value **Yes** is selected for the **Create Azure Run As account** option, and click the **Create** button.
8. While Azure creates the Automation account, you can track the progress under **Notifications** from the menu.

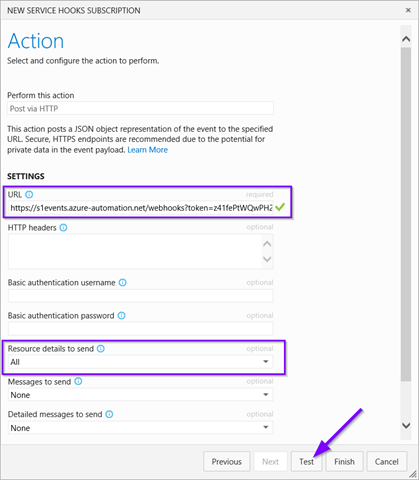
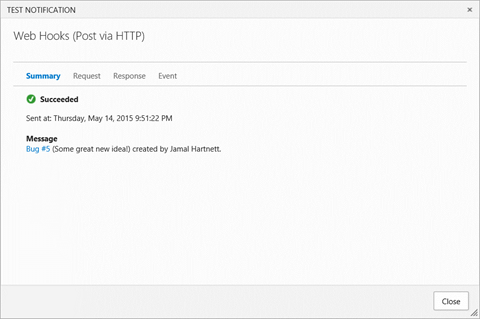
#### Task 2: Test azure automation account

1. In the Azure Portal, open the Automation account created earlier.
2. Click on the **Runbooks** tile to open the list of runbooks.
3. Select the **AzureAutomationTutorialScript** runbook and then click **Start** to start the runbook. You will receive a prompt verifying you wish to start the runbook.
4. A [runbook job](https://docs.microsoft.com/en-us/azure/automation/automation-runbook-execution) is created, the Job blade is displayed, and the job status displayed in the **Job Summary** tile.
5. The job status will start as *Queued* indicating that it is waiting for a runbook worker in the cloud to become available. It will then move to *Starting* when a worker claims the job, and then *Running* when the runbook actually starts running.
6. When the runbook job completes, we should see a status of **Completed**.  
   
7. To see the detailed results of the runbook, click on the **Output** tile.
8. In the **Output** blade, you should see it has successfully authenticated and returned a list of all resources available in the resource group.
9. Close the **Output** blade to return to the **Job Summary** blade.
10. Close the **Job Summary** and the corresponding **AzureAutomationTutorialScript** runbook blade.

#### Task 3: Create a webhook and verify in VSTS

* 1. 1. Create new webhook.  We now have a Webhook that allows us to call that Runbook from VSTS (or really anywhere for that matter). **NOTE**:  Make sure you save the URL, because once you click Create, there is no way to retrieve it again.
  2. 2. Browse to your VSTS account and the project you wish to integrate with.  From the project home page, click on the gear in the top-right corner.
  3. 3. From the control panel page, click on the Service Hooks tab. Then create a new Web Hook subscription.  
      

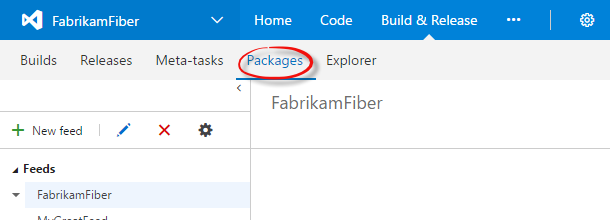
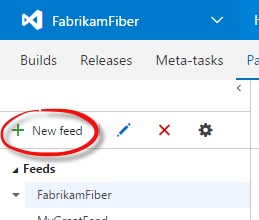
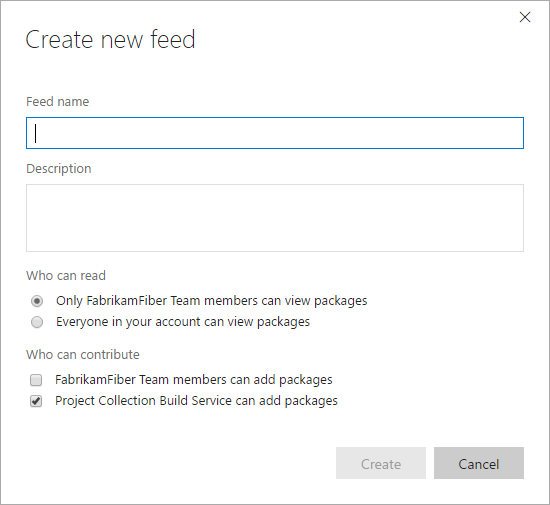
4. Set the Trigger value to Work item created. You can build integrations off any of the events listed but we’ll use this one for our purposes. 

* 1. In the URL field, enter the Webhook URL created from Step #1.  You have the option to configure additional settings, but for our purpose we’ll configure it as shown below  Then click Test. 
  2. When you click Test, it will send test data to our Azure Automation Runbook via the Webhook.  Assuming everything is OK, you should get a success message like below.

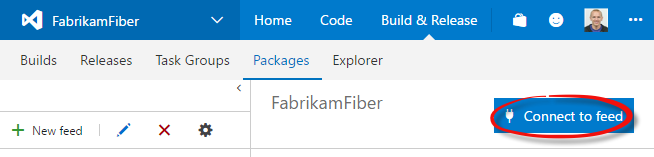
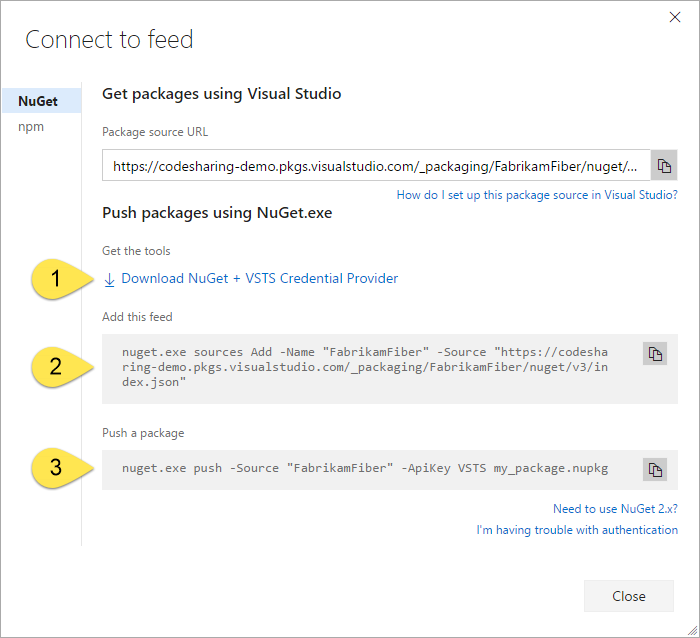
Exercise 2: Use and create packages

#### Note: You must have installed the package management extension from the VSTS markeplace to get this started.

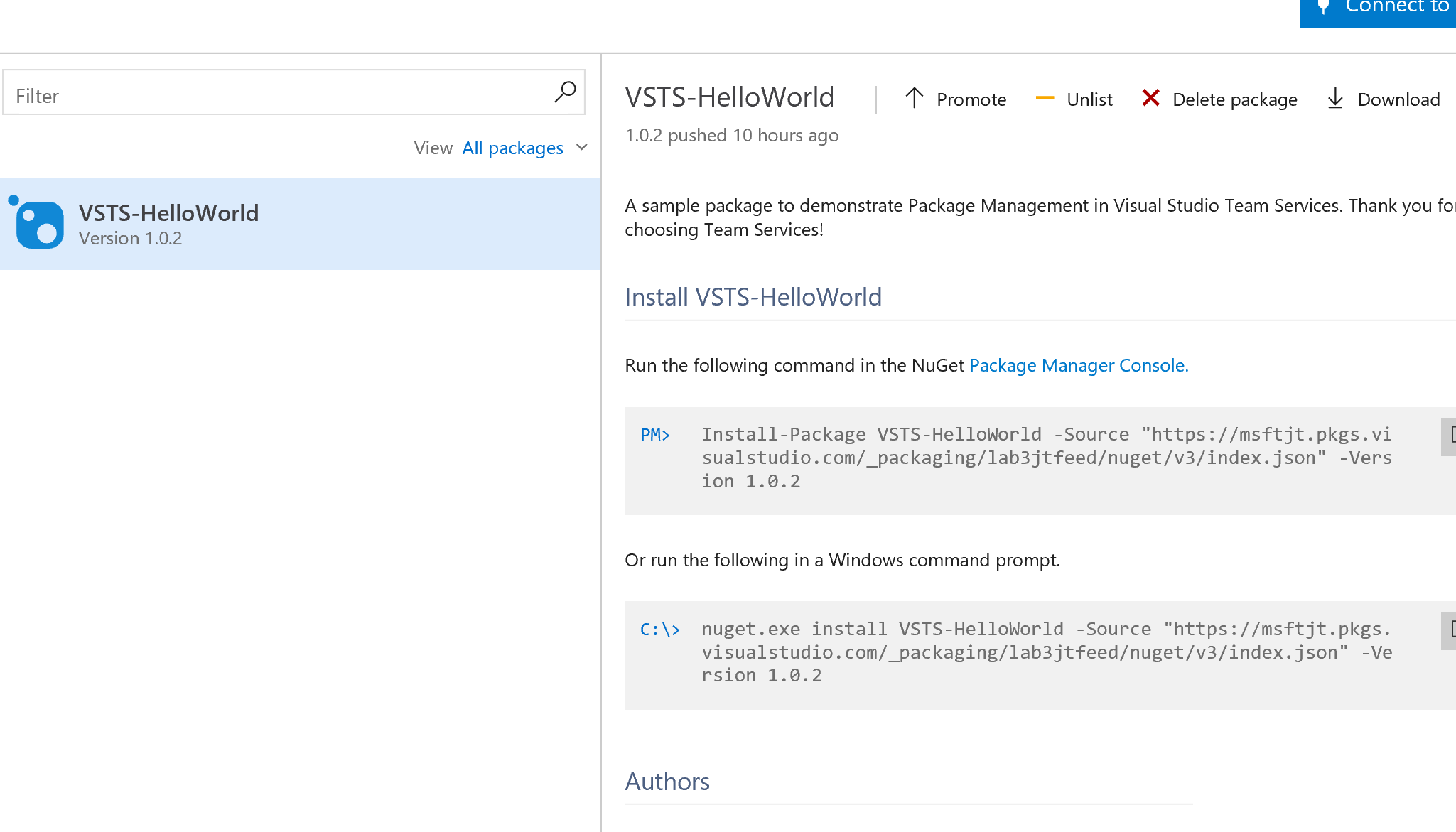
#### Task 1: Create a feed

* 1. 1. Navigate to the **Package** hub in the **Build & Release** hub group:
  2. 
  3. 2. At the top of the left nav bar, select **New feed**:
  4. 
  5. 3. In the dialog:
* Give the feed a name and a description (optional).
* Choose who can read and contribute (or update) packages in your feed. The default selections give Team Build access so that continuous integration builds can add and update packages.
* When you're done, choose **Create**.
  1. 
  2. You can change these settings later by [editing the feed](https://www.visualstudio.com/en-us/docs/package/feeds/edit-feed).

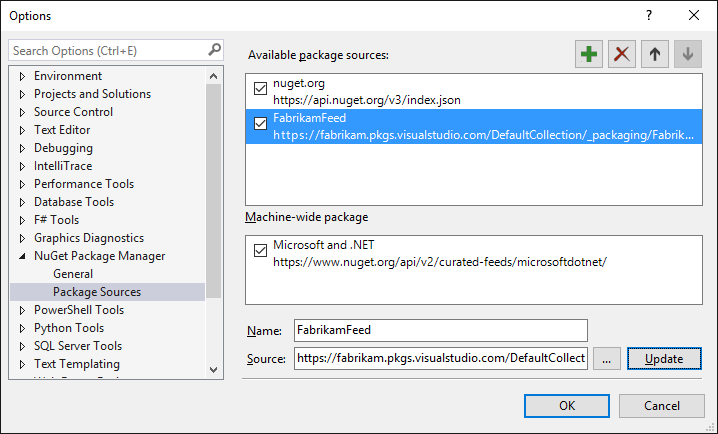
#### Task 2: Publish a package

* 1. 1. Navigate to your feed. Then, select **Connect to feed**:
  2. 
  3. Next, follow steps 1 and 2 to get the tools and add the feed to your local NuGet configuration. Note your dialog will look different depending on what you’ve named your feed. Just download the tools, and then run #2 from the command line.
     1. a. Download the tools, extract them
     2. b. Open a command line, and navigate to nuget.exe
     3. c. Paste the #2 command after you call nuget.exe
     4. d. Skip #3
  4. 

1. 3. Run these commands from the nuget command prompt
2. nuget.exe install VSTS-HelloWorld -ExcludeVersion
3. nuget.exe push -Source {NuGet package source URL} -ApiKey key VSTS-HelloWorld\VSTS-HelloWorld.nupkg

4. Refresh your packages screen to verify its been published.

#### Task 3: Add the feed to your NuGet configuration in Visual Studio

1. On the **Tools** menu, select **Options...**.
2. Expand **NuGet Package Manager** and select **Package Sources**.
3. Click the **green plus** in the upper right corner.
4. At the bottom of the dialog, enter the feed's name and the URL you got in the last step.
5. Select *Update* and then *OK*.
   1. 

#### Task 4: Consume your feed in Visual Studio

1. Find your project in Solution Explorer.
2. Right-click **References**.
3. Select **Manage NuGet Packages...**.
4. In the **Package source** dropdown, select your feed.
5. Look for your package in the list.
   1. 