Workshop on Chocolatey

Summary

In this workshop, you will learn both simple and advanced scenarios for Chocolatey. You will see that Chocolatey can manage anything software-related when it comes to Windows. Here are some of the

things you will learn in this workshop:

- General Chocolatey use
- General packaging
- Customizing package behavior at runtime (package parameters)
- Extension packages
- Custom packaging templates
- Setting up an internal Chocolatey.Server repository
- Adding and using internal repositories
- Reporting
- Advanced packaging techniques when installers are not friendly to automation

You can complete this workshop with either Chocolatey open source (FOSS) or with Chocolatey for Business (C4B). There are sections that specifically apply to C4B. You can skip those sections if they don't apply (or set up a trial - contact sales to take a closer look).

Requirements

- A Windows or macOS machine.
- At least 40GB of free space.
- Recommend 8GB+ of RAM, or as much as possible.
- IN PERSON WORKSHOP: Will need to be able to accept a USB key to transfer setup files.
- NOT IN PERSON WORKSHOP: A decent internet connection as you will need to download a 4GB image. If you are completing this workshop in person with the Chocolatey team, it's likely you are completing an offline workshop. Otherwise you are going to need internet to download everything.

Terminology

- Package - in Chocolatey-speak, package is strictly a Nupkg file. Binaries and installers are referred to as software or binaries. This reduces confusion.
- C4B - you see this next to some exercises. This is the short form of Chocolatey for Business.
- MSP - Managed Service Provider. Also a licensed edition of Chocolatey that has less features and support than C4B, but has a price point that works well for MSP organizations.
- See Notes

Types of Packages

- Installer Package - contains an installer (everything in template is geared towards this type of package)
- Zip Package - downloads or embeds and unpacks archives, may unpack and run an installer using `Install-ChocolateyInstallPackage` as a secondary step.
- Portable Package - Contains runtime binaries (or unpacks them as a zip package) - cannot require administrative permissions to install or use

- Config Package - sets config like files, registry keys, etc
- Extension Package - Packages that add PowerShell functions to Chocolatey - https://chocolatey.org/docs/how-to-create-extensions
- Template Package - Packages that add templates like this for `choco new -t=name` - https://chocolatey.org/docs/how-to-create-custom-package-templates
- Other - there are other types of packages as well, these are the main package types seen in the wild

Exercises

Some of these exercises require a license for the commercial version of Chocolatey. They can be completed with a trial version, but may require pressing enter a few times (and repeating the command if the trial decides not to let Package Builder finish).

If you are completing this lab for FOSS (open source), simply skip those exercises/steps that are indicated by either `C4B`, `MSP`, or `Licensed`.

Notes

- OFFLINE - Offline workshop typically means you are attending the workshop in person somewhere and all materials will be provided so you don't need to download anything.
- PHYSICAL - using a physical machine instead of the provided VM.

Exercise 0: Setup

It's preferred that you perform all of this exercise from a Vagrant image, but you can follow along with a physical Windows box. With Vagrant, you will need either VirtualBox, VMWare Fusion, or Hyper-V for the box provider.

Vagrant Setup

1. Ensure you have a recent version of Vagrant. It is suggested you have at least 1.8.x for linked clones which makes Windows VMs come up lightning quick. Windows machine - `choco install vagrant -y` (then `refreshenv`). OFFLINE: You should find this in the chocolatey-workshop/setup folder that was provided in the USB key.
2. Pre-download the vagrant box we will be using - `vagrant init ferventcoder/win2012r2-x64-nocm` (this is a 4GB box, about 8GB unpacked). OFFLINE: You should find this in the vagrant_boxes folder that was copied to your system. If you didn't get this set as part of the setup.ps1, you will need to run something like `vagrant box add --name ferventcoder/win2012r2-x64-nocm ./path/to/vagrant.box`
3. VirtualBox - If you are completing the workshop with VirtualBox, ensure you have VirtualBox 5 or 5.1 installed. Windows install is `choco install virtualbox -y`. OFFLINE: You should find this in the chocolatey-workshop/setup folder.
4. VirtualBox / MacOS - After you install VirtualBox, open VirtualBox and then Preferences. Go to Input -> Virtual Machine and remap the Host Key to Right Command or Right Alt instead of the default. We need the default to be for the Windows key.
5. OFFLINE WORKSHOP: If we've had you copy files for offline use of this workshop, copy the packages folder into the `demo/packages` () folder. Also copy files from the `downloads` folder to `resources/installers`.

6. OFFLINE: Go to `shell/InstallChocolatey.ps1` and set `$installLocalFile = $true` on line 1.
7. C4B: Place the license you received (by email or fileshare) in `demo/resources/licenses`. Make sure it is named `chocolatey.license.xml`. OFFLINE: This should already be placed, but ensure that it is there.
8. C4B TRIAL: If you have a trial license, put the `chocolatey.extension` package into the packages folder.
9. In your command shell, make sure you are in the `demo` subfolder here. Run `vagrant up` (or `vagrant provision` if already running).
10. Check for errors. Once it finishes, run `vagrant reload` to get copy paste working. Something in Virtualbox/vagrant doesn't apparently set things up properly on first provision.

Troubleshooting: If you run into a UUID issue, you may want to vagrant destroy, then find the `.vagrant` folder in the demo folder or a folder just above and remove that.

In the VM / physical system for workshop completion

All the rest of these commands will be done inside the Vagrant box (or box you are using for this workshop).

1. **NOTE:** If you have placed the license file, you will see a message that looks like an error not finding licensed code. We are going to do that at the end of this, so bare with the issues.
2. Run the following: `choco source add -n local -s c:\vagrant\packages --priority 1`. PHYSICAL: This is likely to be `choco source add -n local -s c:\<directories>\chocolatey-workshop\demo\packages --priority 1`.
3. OFFLINE: Run the following: `choco source disable -n chocolatey`
4. Licensed: Ensure that there is a file at `C:\ProgramData\Chocolatey\license` named `chocolatey.license.xml`. If not, you missed a step above, please manually set the file so you get a warning about being licensed without the licensed extension when you run `choco -v`.
5. Licensed: Install the licensed edition of Chocolatey:
    - Type `choco install chocolatey.extension -y` (ensure you added the nupkg for the packages folder if running a trial)
    - If you get curious, check out `choco source list`.
6. Run the following commands: ~~~sh choco config set cacheLocation 'c:\programdata\choco-cache' ~~~
7. C4B: Run the following commands: ~~~sh choco config set virusScannerType VirusTotal choco feature enable -n virusCheck choco feature enable -n allowPreviewFeatures choco feature enable -n internalizeAppendUseOriginalLocation choco feature enable -n reduceInstalledPackageSpaceUsage ~~~
8. Install .NET Framework 4.5.2 - `choco install dotnet4.5.2 -y`
9. (***HOST***) Run `vagrant reload` to reboot the machine. Okay, there are a few commands you might run from the host machine, typically those are vagrant commands. ***PHYSICAL***: Restart your machine
10. Install the latest GUI - `choco install chocolateygui --pre -y` - this may error with 1603 if you have not rebooted the machine after installing .NET Framework 4.5.2.
11. Install/Upgrade Launchy, Notepad++, Baretail, and Git - `choco upgrade launchy notepadplusplus baretail git -y`.
12. Add the PowerShell profile - type `Set-Content -Path $profile -Encoding UTF8 -Value ""`
13. Type `Write-Host $profile`.
14. Type `start $profile`. Add the following content, then save and close the file: ~~~powershell

$ChocolateyProfile = "$env:ChocolateyInstall\helpers\chocolateyProfile.psm1" if (Test-Path($ChocolateyProfile)) { Import-Module "$ChocolateyProfile" } ~~~

15. Type `. $profile` to reload the PowerShell profile.
16. Create a folder for packages if it doesn't already exist at "c:\packages" - `New-Item 'c:\packages' -ItemType 'Directory' -Force`
17. Navigate to the packages folder. All commands from here will be in that packages folder.

Exercise 1: Install Visual Studio Code

1. Call `choco install vscode -y`
2. Note the message "Environment Vars have changed".
3. Type `code`. Notice that it errors.
4. Type `refreshenv`.
5. Type `code`. Note that it opens Visual Studio Code.

Exercise 2: Create a package the old fashioned way

This is meant to be an exploratory exercise and intentionally doesn't doesn't provide much direction. Most other exercises contain all steps and are very reflective.

1. Download Google Chrome from https://dl.google.com/tag/s/dl/chrome/install/googlechromestandaloneenterprise64.msi and https://dl.google.com/tag/s/dl/chrome/install/googlechromestandaloneenterprise.msi. OFFLINE: These files should be in C:\vagrant\resources\installers.
2. From a command line, call `choco new googlechrome`
3. Go into the googlechrome folder and read the readme, look through the files that were created and try to create a package just using the 64 bit Google Chrome MSI.
4. Run `choco pack`
5. Install the package using Chocolatey - `choco install googlechrome -y -s .`

Note that first time packaging this kind of throws you into the thick of it to see if the information provided is enough to move forward.

Exercise 3: Create a package with Package Builder CLI (C4B)

1. Let's create that GoogleChrome package again.
2. Run `choco new --file googlechromestandaloneenterprise.msi --file64 googlechromestandaloneenterprise64.msi --build-package --outputdirectory $pwd`
3. Inspect the output.

Exercise 4: Create a package with Package Builder UI (C4B)

Let's start by packaging up and installing Puppet 1. Run PowerShell as an administrator 1. Type `packagebuilder` and hit enter. 1. Go to http://downloads.puppetlabs.com/windows/puppet5/ (URL: http://downloads.puppetlabs.com/windows/puppet5/puppet-agent-5.0.1-x86.msi - SHA256: 1D1D45FBF8134A70EA3A39F42CA070BD6600B2FA9506B186EBABA20D770858B1 / Url64: http://downloads.puppetlabs.com/windows/puppet5/puppet-agent-5.0.1-x64.msi - SHA256: 992FD379F60C6D57E9E819CFE7EAD423D1C8B547A6994113ECB96A8C0EE6227D). 1. In the interface that comes up, let's put in the Url and 64-bit Url. 1. Also pass the SHA for verifying the file is what we hope. 1. Click the box next to "Don't embed (don't include software binaries in package)?"

1. Click on Nuspec Information tab. 1. In id, insert "puppet-agent". 1. Click Generate 1. Note that it creates a full package. 1. Open up the packaging files in code. 1. Open the nuspec file. * Note how auto-detection filled out some of the fields here * Optionally we can remove some of the commented sections to tidy this file up and provide more information. 1. In the chocolateyInstall.ps1, note that it captured all MSI properties and prepared a fully ready to go installation. * Also note how it created nice packaging. * Optionally we can remove some of the comments and areas we don't need to tidy this up. 1. Right click on nuspec and select "Compile Chocolatey Package..." / type `choco pack` from that directory. 1. Copy the resulting file to the parent `packages` directory 1. Call `choco install puppet-agent -s . -y` (this tells Chocolatey to install from the local source location ".", which is current directory in both PowerShell.exe and Cmd.exe)

Exercise 5: Create a package with Package Builder (Right Click) (C4B)

1Password

1. Download 1Password from this link - https://d13itkw33a7sus.cloudfront.net/dist/1P/win4/1Password-4.6.0.598.exe (ensure you unblock the file) OR OFFLINE: Find the file in `resources/installers` folder.
2. Right click on the file and choose "Create Chocolatey Package w/out GUI" - **NOTE**: This may error if UAC is on - if so, choose `Create Chocolatey Package...` instead and just click Generate when it comes up.
3. Inspect the output.
4. Let's install this package - `choco install 1password -s . -y --dir c:\programs\1password` (from the working directory where the nupkg is located).

Charles Proxy

1. Download Charles Proxy (both 32 and 64 bit) - https://www.charlesproxy.com/download/ OR OFFLINE: Find the file in `resources/installers` folder.
2. Right click on the 32 bit download and choose "Create Chocolatey Package..."
3. Add the 64bit one into the field.
4. Click generate.
5. Inspect the output.
6. Install this package with `choco install charles -s . -y` (from the working directory where the nupkg is located)

7-Zip EXE

1. Download 7zip (EXE version) - http://www.7-zip.org/download.html (just the 64bit version) OR OFFLINE: Find the file in `resources/installers` folder.
2. Right click and choose "Create Chocolatey Package..."
3. Click Generate.
4. Inspect the output.
5. Open the TODO file that is generated and read over it.
6. Note that it doesn't necessarily figure out the silent arguments.
7. Add the proper silent arguments in the install script.
8. Right click on the 7zip nuspec and select "Compile Chocolatey Package..."

7-Zip MSI

1. Download 7zip (MSI version) - http://www.7-zip.org/download.html (just the 64bit version) OR OFFLINE: Find the file in `resources/installers` folder.
2. Right click and choose "Create Chocolatey Package..."
3. Click Generate.
4. Inspect the output. Note that it determines everything nicely.
5. Right click on the 7zip nuspec and select "Compile Chocolatey Package..."

Exercise 6: Create all the packages (C4B)

1. Type `packagebuilder`.
2. Change output directory to add "programs" to the path (just to keep things separate). `C:\packages\programs` if you are in the packages folder.
3. Click on the Programs and Features tab.
4. Click `Generate` in that tab.
5. Note the output.

6. Look at package folders that didn't generate a nupkg.

   OR

7. `choco new --from-programs-and-features --build-package --outputdirectory programs`

8. Note the output.
9. Look at package folders that didn't generate a nupkg.

Exercise 7: Set up a local Chocolatey.Server

1. ***HOST***: Start with `vagrant reload` to clear pending reboots.
2. Start the Windows Update service - `Get-Service wuauserv | Set-Service -StartupType Automatic -Passthru | Start-Service`
3. Install KB2919442 - `choco install KB2919442 -y` (note this may take a long time to install)
4. ***HOST***: Run `vagrant reload`.
5. Install KB2919355 - `choco install KB2919355 -y` - this one or the other Windows update takes a ***very*** long time to install, just be patient and let it complete.
6. ***HOST***: Run `vagrant reload`.
7. Run `choco install dotnet4.6.1 -y`
8. Stop the Windows Update service - `Get-Service wuauserv | Set-Service -StartupType Disabled -Passthru | Stop-Service`
9. ***HOST***: You guessed it, one more time - `vagrant reload`
10. Ensure IIS and Asp.NET are installed
    - `choco install IIS-WebServer -y --source windowsfeatures`
    - `choco install IIS-ASPNET45 -y --source windowsfeatures`
11. `choco install chocolatey.server -y`
12. Follow instructions at https://chocolatey.org/docs/how-to-set-up-chocolatey-server
13. Go to http://localhost, verify the setup, look at the password.
14. Add this repository to your default sources. Call it `internal_chocolatey` - try `choco source -?` to learn how. Ensure `http://localhost/chocolatey` is the source location you use.

We are moving towards updating the base image to speed this bit up so that there is not the waiting

around on it.

Exercise 8: Push a package to a Chocolatey Server

1. Run `choco search -s http://localhost/chocolatey`
2. In the folder where we've generated packages, let's find 1Password nupkg.
3. Run `ls` to verify that there is a 1Password nupkg in the directory. If not, you need to navigate to that folder.
4. Run `choco push -s http://localhost -k chocolateyrocks` (note the push is different than the query url) - we only need to specify path to the nupkg if there is more than one in the current directory.
5. Run `choco search -s http://localhost/chocolatey`. Run `choco search -s internal_chocolatey` and note the output should be the same.
6. Note that the package is available.

Exercise 9: Upgrade a package

1. Download an updated 1Password from this link - https://d13itkw33a7sus.cloudfront.net/dist/1P/win4/1Password-4.6.1.617.exe OR OFFLINE: Find the file in `resources/installers` folder.
2. Use any method for creating packages to generate the packaging for this upgrade. **NOTE**: When you do this, you may need to rename the existing 1Password package folder first.
3. Instead of using Package Builder, you can instead download the file into the tools directory of the current package, edit the nuspec version, delete the previous installer exe, and update the chocolateyInstall.ps1 to point to the new installer and then compile the package.
4. Push this updated package to the Chocolatey server.

Exercise 10: Install package from Internal Repository

1. Run `choco search 1password -s internal_chocolatey` (or whatever name you passed)
2. Run `choco search 1password -s internal_chocolatey --all-versions`. Note the output.
3. Run `choco search 1password --detailed`. Note the output.
4. Run `choco upgrade 1password -s internal_chocolatey -y`

Exercise 11: Reporting

1. Run `choco list -lo --include-programs`
2. Note the output.
3. Run `choco outdated`
4. Note the output.

Exercise 12: Package Synchronizer - Automatic Sync (Licensed)

1. Run `choco list -lo --include-programs`.
2. Go to `C:\ProgramData\Chocolatey\lib`. Note the 1password package.
3. Rename the `C:\ProgramData\Chocolatey\license` folder to `licensed`. This will unlicense Chocolatey.
4. Go to Programs and Features.
5. Manually uninstall 1Password.
6. Run `choco list -lo --include-programs`. Note if 1password package is still there.
7. Rename the `C:\ProgramData\Chocolatey\licensed` folder to `license`. This will license Chocolatey.

8. Run `choco list -lo --include-programs`. Note if 1password package is still there.
9. Look for a lib-synced folder in `C:\ProgramData\Chocolatey`.
10. Note the contents.
11. You may need to reset some features due to running unlicensed Chocolatey. Run the following commands: ~~~sh choco feature enable -n virusCheck choco feature enable -n allowPreviewFeatures choco feature enable -n internalizeAppendUseOriginalLocation choco feature enable -n reduceInstalledPackageSpaceUsage ~~~

Exercise 13: Package Synchronizer - Choco Sync (C4B)

1. Go to `C:\ProgramData\chocolatey\.chocolatey` and delete the 7zip folder if it exists. Otherwise delete the 1password folder (these folders will have a version after them).
2. Run `choco list -lo --include-programs`.
3. Note any applications not being managed as Chocolatey packages.
4. Run `choco sync` - **NOTE**: If this errors, make sure you've turned on allowPreviewFeatures from exercise 0 to allow this feature to work.
5. Note how it relinks 7zip or 1password to an existing installed package. This is recreating a lost link.
6. Note that it is syncing with new packages for the rest of the items.
7. Go to the temp directory to see the packaging it created.
8. Run `choco list -lo --include-programs`.
9. Note any applications not being managed as Chocolatey packages.

Exercise 14: Manually Internalize Notepad++ package

1. Follow the instructions at internalize an existing package manually to internalize Notepad++.

Exercise 15: Internalize Visual Studio Code package (MSP/C4B)

1. Run `choco feature list`. Determine if `internalizeAppendUseOriginalLocation` is on. Turn it on otherwise.
2. Call `choco download vscode --internalize --resources-location http://somewhere/internal -s https://chocolatey.org/api/v2/` (literally). You only need to specify source if you've disabled the community repository source.
3. While it is downloading, head into the download folder it created.
4. Open the `download\vscode\tools\chocolateyInstall.ps1` (relative to the current working directory) in Notepad++ or Code.
5. Note the url variable.
6. When it finishes downloading and creating the package, note how that changes.
7. Note how it appended `-UseOriginalLocation` in this case.

Exercise 16: Internalize AdobeReader package (MSP/C4B)

1. Run `choco feature list`. Determine if `internalizeAppendUseOriginalLocation` is on. Turn it on otherwise.
2. Call `choco download adobereader --internalize -s https://chocolatey.org/api/v2/` (you only need the source if you've disabled the community repo source)
3. While it is downloading, head into the download folder it created.
4. Open the chocolateyInstall.ps1 in Notepad++ or Code.
5. Note the url variable.
6. When it finishes downloading and creating the package, note how that changes.
7. Note that there is a files folder that contains the binaries.

8. Note how it has appended `-UseOriginalLocation` to the end of `Install-ChocolateyPackage`.

NOTE: You can also complete this exercise with Chocolatey FOSS by manually internalizing the package, see manually internalizing packages for more information.

Exercise 17: Download Chocolatey and Licensed packages (Licensed)

To have a completely offline install for packaging, you need to remove

1. Run `choco source list` to see your sources.
2. Run `choco source disable -n chocolatey`
3. Run `choco download chocolatey -s https://chocolatey.org/api/v2/`
4. Run `choco download chocolatey.server -s https://chocolatey.org/api/v2/`
5. Run `choco download chocolatey.extension --ignore-dependencies --source https://licensedpackages.chocolatey.org/api/v2/`
6. C4B: Run `choco download chocolatey-agent --ignore-dependencies --source https://licensedpackages.chocolatey.org/api/v2/`
7. Run `choco source disable -n chocolatey.licensed` - **NOTE**: When you have a licensed version of Chocolatey, you are unable to remove this source. It can be disabled though. Also, once this is disabled, you would need your license id as the password you would pass to the licensed source in the previous steps.
8. Push all of these packages to your internal server.
9. You are now using Chocolatey with internal only packages.

Exercise 18: Create an extension package

We are going to create a package that checks for prerequisites prior to the install, such as ensuring at least 3 GB of free space.

1. Run `choco new prerequisites.extension`
2. Delete the `prerequisites.extension\tools` directory.
3. Create an `extensions` directory.
4. Create a file called `prerequisites.psm1` in the extensions directory.

5. Add this to the contents:

```
# Export functions that start with capital letter, others are private
# Include file names that start with capital letters, ignore others
$ScriptRoot = Split-Path $MyInvocation.MyCommand.Definition

$pre = ls Function:\*
ls "$ScriptRoot\*.ps1" | ? { $_.Name -cmatch '^[A-Z]+' } | % { . $_  }
$post = ls Function:\*
$funcs = compare $pre $post | select -Expand InputObject | select -Expand
Name
$funcs | ? { $_ -cmatch '^[A-Z]+'} | % { Export-ModuleMember -Function $_ }
```

6. Create a file `Ensure-ThreeGBs.ps1` and add the following contents:

```
<#
```

```
  .SYNOPSIS
  Ensures that there is at least 3 GB of space left on a machine or it will
not allow installation

  .OUTPUTS
  [String]
#>
function Ensure-ThreeGBs {
  Write-Debug "Running Get-AvailableDiskSpace to determine if there is enough
space for installation."

  $disk = Get-PSDrive C | Select-Object Used,Free
  Write-Host "There is $($disk.Free) available"

  $ThreeGBs = 3221225472
  if ($disk.Free -lt $ThreeGBs) {
    throw "There is less than 3GB of free space left."
  }

  return $disk.Free
}
```

7. Update the nuspec appropriately. Ensure the version is at least `0.0.1`.
8. In the nuspec, change `<file src="tools\**" target="tools" />` to `<file src="extensions\**" target="extensions" />`.
9. Run `choco pack` against the directory with `prerequisites.extension.nuspec`.
10. Copy the nupkg file up to the packages directory.
11. Now head into the 1Password package from exercise and open `tools\chocolateyInstall.ps1`.
12. On line 1, add the following: `Ensure-ThreeGBs`. Save and close.
13. Open up `1password.nuspec` and add a dependency on the prerequisites.extension package (right before `</metadata>`): ~~~xml ~~~
14. Compile the 1password package back up and put it in the folder next to the `prerequisites.extension` nupkg.
15. Run `choco install 1password -s . -y`. **NOTE**: We uninstalled this with auto sync in an earlier exercise.
16. Note in the install how it automatically loads up the prerequisites functions and makes them available without any more work on the part of the installation scripts.

NOTE: Learn more at https://chocolatey.org/docs/how-to-create-extensions.

Exercise 19: Create a package template for MSIs

1. Run `choco new msi.template`.
2. Delete the `msi.template\tools` directory.
3. Create `templates\msi.nuspec.template` and add the following contents: ~~~powershell <?xml version="1.0" encoding="utf-8"?> [[PackageNameLower]] [[PackageName]] (Install) [[PackageVersion]] Original authors [[MaintainerName]] __REPLACE__MarkDown_Okay [[AutomaticPackageNotesNuspec]] [[PackageNameLower]] admin ~~~

4. Create `templates\tools\chocolateyInstall.ps1` and add the following contents:

```
$ErrorActionPreference = 'Stop';
```

```
$packageName= '[[PackageName]]'
$toolsDir   = "$(Split-Path -parent $MyInvocation.MyCommand.Definition)"
$fileLocation = Join-Path $toolsDir 'NAME_OF_EMBEDDED_INSTALLER_FILE'

$packageArgs = @{
  packageName   = $packageName
  unzipLocation = $toolsDir
  fileType      = 'msi'
  file         = $fileLocation
  softwareName  = '[[PackageName]]*' #part or all of the Display Name as you
see it in Programs and Features. It should be enough to be unique
  silentArgs    = "/qn /norestart /l*v
`"$env:TEMP\chocolatey\$($packageName)\$($packageName).MsiInstall.log`"" #
ALLUSERS=1 DISABLEDESKTOPSHORTCUT=1 ADDDESKTOPICON=0 ADDSTARTMENU=0
  validExitCodes= @(0, 3010, 1641)
}

Install-ChocolateyInstallPackage @packageArgs
```

5. Open `msi.template.nuspec` and edit it appropriately. Set the version to `1.0.0`.
6. In the nuspec, change `<file src="tools\**" target="tools" />` to `<file src="templates\**" target="templates" />`.
7. Call `choco pack`.
8. Now we can push this up to our package server.
9. Let's install this template - `choco install msi.template -s internal_chocolatey`.

NOTE: Learn more at https://chocolatey.org/docs/how-to-create-custom-package-templates.

Exercise 20: Create a package from a template

1. Run `choco new bob -t msi`.
2. Head into the bob folder.
3. Note how it does replacements of all of the `[[variables]]`

Exercise 21: Update a packaging template / use custom properties

1. Let's add a new variable.
2. Open `msi.template\templates\tools\chocolateyInstall.ps1`
3. Add the following at the top: `# [[CustomVariable]]`
4. Save and close that file.
5. Open `msi.template.nuspec` and increase the version to `1.0.1` .
6. Close that, package it up and push it up to the server again.
7. Run `choco upgrade msi.template -s internal_chocolatey`.
8. Now run `choco new tim -t msi CustomVariable="Yes"`
9. Note the output in the tim folder that is created.

Exercise 22: Use package parameters

1. Run `choco new packagewithparameters`
2. Remove everything but the nuspec and `tools\chocolateyInstall.ps1`.
3. In the nuspec, take a dependency on `chocolatey-core.extension` version `[1,3)` (which means at least v1, but anything less than v3) and change the version to `0.0.1`.

4. In the `chocolateyInstall.ps1`, delete everything and just add the following:

```
$pp = Get-PackageParameters

if (!$pp['LICENSE']) { $pp['LICENSE'] = 'License123' }
#if (!$pp['LICENSE']) { $pp['LICENSE'] = Read-Host "Please provide LICENSE" }

Write-Warning "LICENSE = '$($pp['LICENSE'])'"
```

5. Package up the package and push it to your internal server.
6. Ensure the chocolatey-core.extension package is up on the internal server as well.
7. Run `choco install packagewithparameters -s internal_chocolatey --params "'/LICENSE:Yes'"` and note the output.

NOTE: Learn more at https://chocolatey.org/docs/how-to-parse-package-parameters-argument.

Exercise 23: Use AutoHotKey for craptastic installers

NOTE: Craptastic is a technical term ;).

1. COMING SOON.