

Windows Software Packaging with Chocolatey

What is Chocolatey?

A close-up shot of a man with a shaved head and sunglasses, sitting in a red leather chair. He has his hands clasped in front of him and a serious expression. The background is dark and out of focus.

Do you want to know

What is Chocolatey?

According to the official Chocolatey website:

“Chocolatey is kind of like apt-get, but for Windows (with Windows comes limitations). **It is a machine level package manager that is built on top of nuget command line and the nuget infrastructure.**”

What is Chocolatey **not**?

What is Chocolatey **not** or what does it not do?

It is not a magic wand or a silver bullet.

It has limitations and pitfalls.

It is a **software packaging and deployment tool. Period.**

It takes effort to do proper software package management. Chocolatey isn't going to solve your software packaging and distribution problem overnight (although it may come close).

Disclaimer: Security and compliance go hand in hand but are **not** the same thing.

Chocolatey can be a huge efficiency-booster for any organization with respect to software package deployment and patching, and can be very secure if used correctly. Unfortunately, like many powerful tools, it can also be very dangerous and has the potential to do incredible damage if wielded incorrectly. **With great power comes great responsibility.**

Ok...so...what's the point?

Open Source Software Deployment

- Open Source Chocolatey and is free (as in beer) to use. Even commercially.
- Packages (which utilize metadata) are built using PowerShell script(s).
- Anything you can do in PowerShell, you can deliver with Chocolatey.
- Packages can be deployed on almost any Microsoft Windows operating system
- Free (as in beer) is a powerful motivator for many organizations.
- So is security compliance. Update **all** installed packages with a single command.
- Public feed is globally-sourced with package maintainers in a moderated community.
- Packages (and binaries) gets scanned by Virus Total and follow a moderation process.
- When was the last time you enjoyed packaging an app and deploying it?

“You’ve never deployed software faster than you will with Chocolatey.”

-Rob Reynolds (creator)

Let's do a history review!

The ~~really old~~ classic way of deploying application software

1. Search for most recent version of the application software you need to deploy
2. Pick (hopefully) the right download.
3. Wait for it to download
4. Maybe verify the checksum (if it's even provided)? Maybe scan for malware.
5. Install it manually (next, next, next anyone?). **Possibly in an OS image?**¹
6. Hope it didn't install any malware
7. Now do it again for EVERY piece of software you need to deploy.
8. Repeat when a new version of the application software is released.

¹Shameless plug for my talk tomorrow about Automating Windows OS Image Creation

The ~~-really old~~ +newer classic way of deploying application software

1. Search for most recent version of the application software you need to deploy
2. Pick (hopefully) the right download.
3. Wait for it to download
4. Maybe verify the checksum (if it's even provided)? Maybe scan for malware.
5. ~~Install it manually (next, next, next anyone?).~~ +Create scripted install. Deploy post-image.¹
6. Hope it didn't install any malware
7. Now do it again for EVERY piece of software you need to deploy.
8. Repeat when a new version of the application software is released.

¹Shameless plug number two for my talk tomorrow about Automating Windows OS Image Creation

The somewhat modern way of deploying application software

1. Search for most recent version of the application software you need to deploy
2. Pick (hopefully) the right download.
3. Wait for it to download
4. Maybe verify the checksum (if it's even provided). Maybe scan for malware.
5. Create a task sequence or package with an end-point management tool
6. Schedule the package installation and deploy to selected systems
7. Hope it didn't install any malware
8. Now do it again for EVERY piece of software you need to deploy.
9. Repeat when a new version of the application software is released.

Deploying application software with Chocolatey

1. To install: `choco install google-chrome -y`
2. To update: `choco upgrade google-chrome -y`
3. To uninstall: `choco uninstall google-chrome -y`
4. To force reinstall: `choco install google-chrome -y -f`

That's it.

So what exactly is a Chocolatey Package?

- Chocolatey packages are known as **nupkg** files, which is a compiled NuSpec (fancy zip file) that knows about package metadata (including dependencies and versioning).
- A Chocolatey package can contain embedded software and/or automation scripts.
- This includes PowerShell scripts (v2+) as well as other types of files including the option of embedding binaries.
- Can be used directly from chocolatey.org/packages or from internal web server.
- Allow easy upgrading, uninstalling, and syncing with Programs and Features.
- Can be used to deploy MSI, Exe, ClickOnce, and many other installer types.
- Even installers with complicated silent installation options, ex. response files.

So what exactly **is** a Chocolatey package?

In short, a zip archive, with a different file extension: `.nupkg`.

Inside this `.nupkg` is a collection of various files. We're going to focus on two files in particular which do most of the heavy lifting through metadata.

The first is the `.nuspec` file, a collection of metadata about the software to be installed including title, version (semver), download URL, author/maintainer, copyright and license info, release notes, etc.

The second is the `chocolatey\Install.ps1` file, which are the actual instructions Chocolatey uses to run an installer from the command-line.

The .nuspec file

Example: PaperCut NG Client

```
<?xml version="1.0" encoding="utf-8"?>
<package xmlns="http://schemas.microsoft.com/packaging/2011/08/nuspec.xsd">
  <metadata>
    <id>vcuPaperCutNG-Client</id>
    <version>18.0.4</version>
    <title>PaperCut NG Client</title>
    <authors>PaperCut Software International</authors>
    <owners>griffithhc</owners>
    <licenseUrl>https://www.papercut.com/kb/Category/Licensing</licenseUrl>
    <projectUrl>https://www.papercut.com/</projectUrl>
    <iconUrl>https://n8felton.files.wordpress.com/2015/06/papercutnewlogo.png?w=1400</iconUrl>
    <requireLicenseAcceptance>true</requireLicenseAcceptance>
    <description>https://www.papercut.com/products/ng/manual/common/topics/feature-sectdesc.html</description>
    <summary>PaperCut NG is a comprehensive print management system designed to monitor and control print
resources.</summary>
    <releaseNotes>https://www.papercut.com/products/mf/release-history/</releaseNotes>
    <copyright>PaperCut Software International</copyright>
    <language>en-US</language>
    <serviceable>true</serviceable>
  </metadata>
</package>
```

chocolateyInstall.ps1

Example: PaperCut NG Client

```
$packageName = 'PaperCut Client Software'
$toolsDir     = "$($Split-Path -Parent $MyInvocation.MyCommand.Definition)"
$fileLocation = Join-Path $toolsDir 'pc-client-admin-deploy.msi'

$zipArgs = @{
    packageName = $packageName
    unzipLocation = $toolsDir
    url = 'https://files.nuget.ts.vcu.edu/packages/ts/PaperCutNGClient.zip'
    checksum = '726A4921529BD40EF3F2EBE676A3493E2A280BCFBEBF45A6C75944CDC2CB15298'
    checksumType = 'sha256'
}

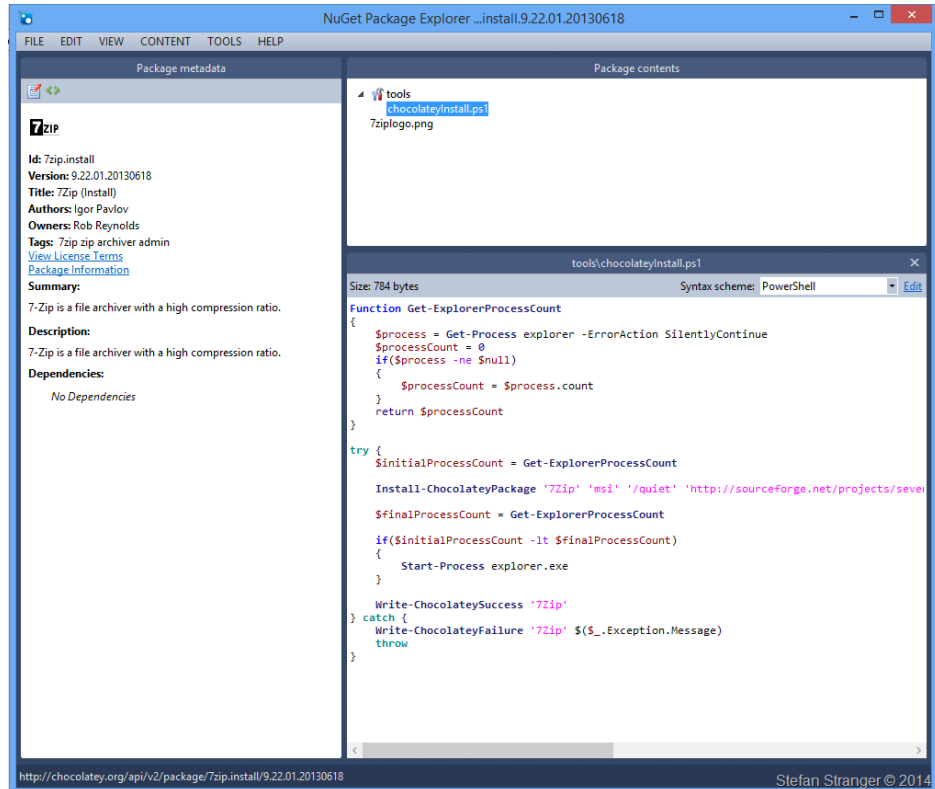
$packageArgs = @{
    packageName = $packageName
    file = $fileLocation
    fileType = 'msi'
    silentArgs = '/qn /norestart ALLUSERS=1'
    validExitCodes = @(0)
}
```

```
Install-ChocolateyZipPackage @zipArgs
```

```
Install-ChocolateyInstallPackage @packageArgs
```

Is there an easier way to build packages?

NuGet Package Explorer



The screenshot displays the NuGet Package Explorer interface for the package `7zip.install`. The window title is "NuGet Package Explorer ...install.9.22.01.20130618".

Package metadata:

- 7ZIP**
- Id:** 7zip.install
- Version:** 9.22.01.20130618
- Title:** 7Zip (Install)
- Authors:** Igor Pavlov
- Owners:** Rob Reynolds
- Tags:** 7zip zip archiver admin
- [View License Terms](#)
- [Package Information](#)

Summary:
7-Zip is a file archiver with a high compression ratio.

Description:
7-Zip is a file archiver with a high compression ratio.

Dependencies:
No Dependencies

Package contents:

- tools
- chocolateyinstall.ps1
- 7ziplogo.png

The `tools\chocolateyinstall.ps1` file is selected, showing the following PowerShell script:

```
Size: 784 bytes Syntax scheme: PowerShell Edit
Function Get-ExplorerProcessCount
{
    $process = Get-Process explorer -ErrorAction SilentlyContinue
    $processCount = 0
    if($process -ne $null)
    {
        $processCount = $process.count
    }
    return $processCount
}

try {
    $initialProcessCount = Get-ExplorerProcessCount
    Install-ChocolateyPackage '7Zip' 'msi' '/quiet' 'http://sourceforge.net/projects/sev...'
    $finalProcessCount = Get-ExplorerProcessCount
    if($initialProcessCount -lt $finalProcessCount)
    {
        Start-Process explorer.exe
    }
    Write-ChocolateySuccess '7Zip'
} catch {
    Write-ChocolateyFailure '7Zip' $('$_Exception.Message')
    throw
}
```

Footer: <http://chocolatey.org/api/v2/package/7zip.install/9.22.01.20130618> Stefan Stranger © 2014

What about security?

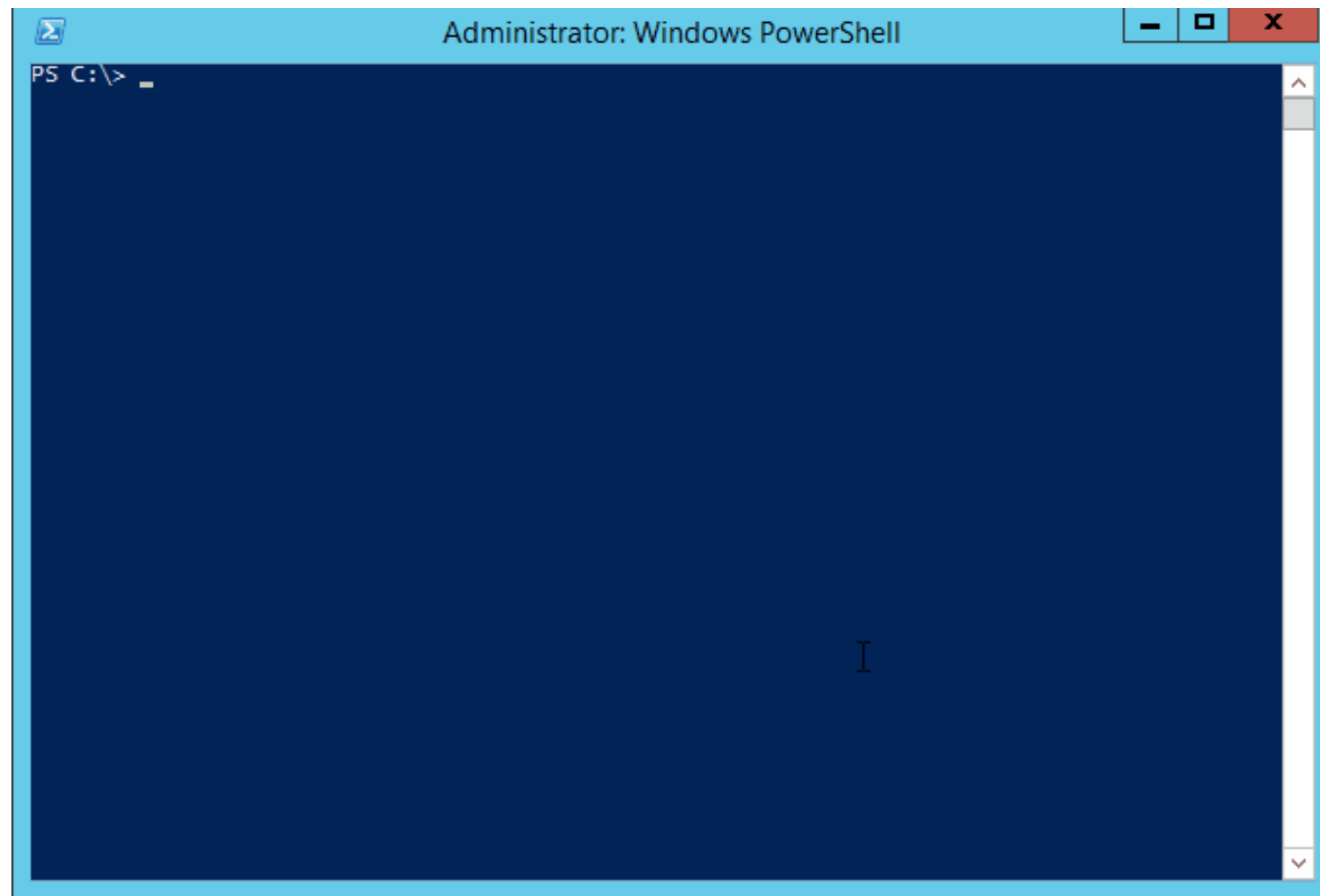
Why you should “internalize” packages

In short, there is no reason you can't use the community repository of Chocolatey packages. However, bear in mind that packages are created and maintained by people from around the world and while the efficiency gained from effectively crowd-sourcing package creation is awesome, it only takes one slip-up for a bad actor to gain a foothold inside your network.

While the most popular packages on the community repository (chocolatey.org) are safe to use, assuming moderators have approved them, you should still review the package code, check virus-total scan results (not only for the package but also the binary), and test before deploying en masse. Repeat with new versions as necessary.

Internalizing packages can be easy with the right tool. It can also save you some hassle in the long run. In fact, internalization is a feature of the paid edition, Chocolatey for Business.

What does Chocolatey look like in action?



Hosting Chocolatey packages internally

Requires a NuGet Server (web server) to serve .nupkg files from a feed/repo.

Use same web server can be used to serve binaries, or a SMB/CIFS share.

These commercial-off-the-shelf software solutions come with a NuGet server built in: Inedo ProGet, JFrog Artifactory, Sonatype Nexus

I'm going to demonstrate how to use ProGet inside your own network to host your internal-only Chocolatey packages for consumption by end-point management solutions and directly via scripts, command-line, and GPOs.

DEMO TIME

End-Point Management Software Platforms

Chocolatey can be used with end-point management, deployment, and orchestration platforms such as:

- Microsoft Deployment Toolkit (MDT)
- System Center Configuration Manager (SCCM)
- LANDesk Management Suite
- Dell KACE
- and many others...

Configuration Management Platforms

Such as:

- Puppet
- Chef
- Ansible
- Salt
- PowerShell DSC
- and again, many others.

Puppet

```
include chocolateypackage { 'git':  
  ensure    => latest,  
  provider => 'chocolatey',  
  source    => 'https://my.internal.repository/api/v2',  
}
```

Chef

```
chocolatey_package 'git' do
  action :install
  source 'https://my.internal.repository/api/v2'
end
```

Ansible

```
win_chocolatey:
```

```
  name: git
```

```
  source: https://my.internal.repository/api/v2/
```

What about GPOs?

Deploying Chocolatey Packages with GPOs

1. Create a new GPO
2. Create a startup script using Powershell that looks like this:

```
choco install package -s https://my.internal.repository/nuget -y
```

or...

Use an array

```
$ChocoPackages = @("googlechrome","adobereader","notepadplusplus.install","7zip.install")

foreach($Package in $ChocoPackages) {
    try {Invoke-Expression "cmd.exe /c choco install $Package -y" -ErrorAction Stop}
    catch {Throw "Failed to install $Package" }
}
```

Q&A

Hands-On Workshop