INFN GENOVA

TUTORIAL DAYS 2022

Installazione e package management di sistemi Windows

E.Salvo

Sommario

TEORIA

Sommario

PRATICA

Esigenze di setup dei client Windows

- **Configurazione delle macchine**:
  - uniforme e replicabile;
  - flessibile;
    (a seconda della destinazione d'uso e del tipo di hardware)
  - automatizzata, anche senza l’uso di GPO;
    (se alcune macchine non possono essere messe in dominio, ad esempio alcuni notebook).

Lo scopo è garantire maggiore controllo, efficienza e sicurezza.

In passato, l'installazione di macchine Windows era gestita tramite **imaging**, un'operazione laboriosa. Oggi, **il processo è completamente automatico** e prevede varie fasi, eseguibili anche manualmente e separatamente, che consentono di installare e configurare automaticamente macchine di diverso tipo e con diverse esigenze.

Tecnologie scelte e processo di installazione

Il sistema di netboot preesistente era configurato per installare solo macchine Linux:

- basato su **PXELinux,** poteva essere inadatto a un'installazione Windows
  (p.e. era limitato riguardo la dimensione dell'immagine di installazione).

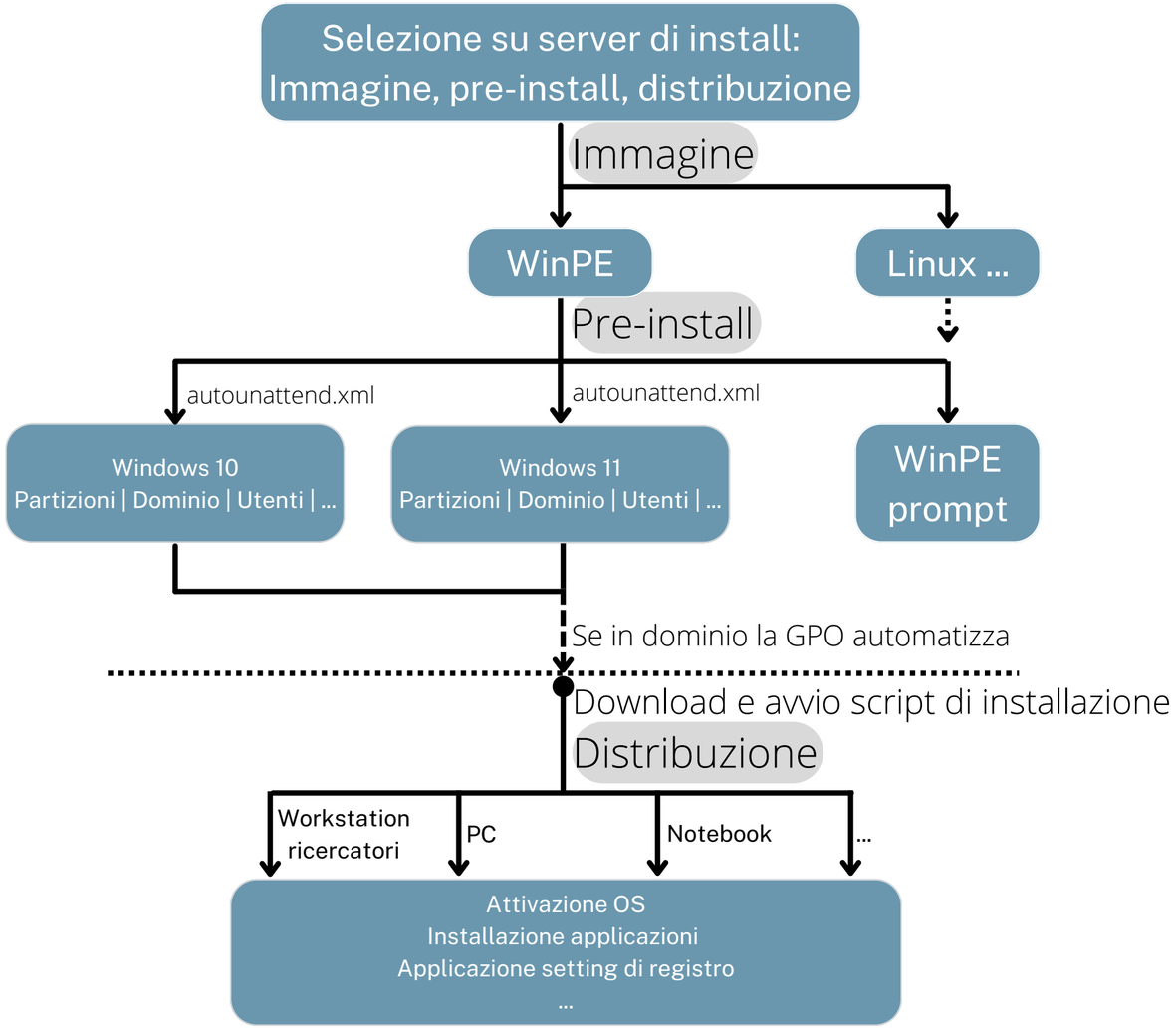
Il nuovo sistema è basato su **iPXE**, che presenta alcuni vantaggi:

- supporta dimensioni di immagini maggiori
- utilizza il protocollo https anziché tftp, più veloce e con migliore correzione dell'errore

iPXE carica WinPE, che, a seconda della configurazione di **pre-install** e di **distribuzione** scelte sul server di install, sceglierà se e come lanciare il setup (tramite un autounattend.xml generato dal server stesso), variando ad es. il partizionamento, l'edizione (Pro, Enterprise...) o l'inserimento in dominio.

In post-install, viene scaricato (se in dominio automaticamente, tramite GPO) un sistema di script (in gran parte basati su **Chocolatey**), che standardizzano applicazioni installate e impostazioni secondo la **distribuzione** scelta, e rendono il sistema pronto all'uso. Questo passo è anche eseguibile manualmente (ad es. su un notebook preinstallato).

Tecnologie scelte e processo di installazione



Cos'è Chocolatey

Un **sistema di gestione di pacchetti** (ispirato da yum e apt) che si basa internamente su nupkg (ispirato da rpm) per l'installazione e l'aggiornamento di programmi e funzionalità in modo automatizzato sui sistemi Windows.

Il suoi principali punti di forza sono:

- L'utilizzo delle librerie di nuget (l'API che gestisce nupkg).
- L'estendibilità tramite moduli powershell
- Il community repository, sempre aggiornato, che contiene quasi ogni applicativo di libero uso (attenzione alle licenze!).

Pacchetti Chocolatey

Pacchetto nupkg:

- è composto da una serie di **script** e **resource** che concorrono a **gestire** (installare e disinstallare) il software pacchettizzato.
- diverso da pacchetti rpm o simili in ambiente Linux:
  - I pacchetti nei repository pubblici contengono quasi sempre solo gli script di gestione, che recuperano l'installer vero e proprio scaricandolo dalla rete, di solito su hosting di terze parti. Questo porta a un **repository leggero**, proprio perché gestisce quasi esclusivamente meta-pacchetti.
  - Il mantenimento è dunque pressoché indolore specialmente in caso di installer msi, molto comuni e standardizzati.
- A meno di cambiamenti radicali nell'installer, gli script rimangono sostanzialmente invariati e va sostituito solo l'installer.

Licenze

Chocolatey è disponibile con varie licenze.

Licenza gratuita:

- liberamente utilizzabile in contesto aziendale

Estendibile con moduli powershell

Licenza business (lista non esaustiva):

- Download di pacchetti (senza doverli effettivamente installare),
- Ripacchettizzazione automatica, utile alla creazione di un repository locale.
- Software installati tracciati anche se non vengono rimossi tramite chocolatey (ricostruisce dinamicamente i metadati dei pacchetti installati)
- Importazione delle applicazioni installate e disinstallazione di esse tramite chocolatey

Installazione di chocolatey sulla macchina client

Solitamente si usa il **comando powershell**, che scarica lo script di installazione e lo esegue:
(chiaramente, va eseguito come amministratore della macchina)

```
Set-ExecutionPolicy Bypass -Scope Process -Force;
[System.Net.ServicePointManager]::SecurityProtocol =
[System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object
System.Net.WebClient).DownloadString('https://community.chocolatey.org/install.ps1'))
```

Oppure la **versione per cmd.exe**:

```
@"%SystemRoot%\System32\WindowsPowerShell\v1.0\powershell.exe" -NoProfile -InputFormat
None -ExecutionPolicy Bypass -Command "
[System.Net.ServicePointManager]::SecurityProtocol = 3072; iex ((New-Object
System.Net.WebClient).DownloadString('https://community.chocolatey.org/install.ps1'))"
&& SET "PATH=%PATH%;%ALLUSERSPROFILE%\chocolatey\bin"
```

Comandi principali di gestione dei pacchetti

`choco install <pacchetti>`

Installa il/i pacchetti specificati. Se un pacchetto è già installato, anche una versione precedente a quella presente nei repository, non verrà compiuta alcuna azione

`choco uninstall <pacchetti>`

Rimuove il/i pacchetti specificati

`choco upgrade <pacchetti>`

Installa il/i pacchetti specificati. Se un pacchetto è già installato, ma la versione sul repository è più recente, verrà eseguito lo script di installazione del pacchetto presente sul repository, aggiornandolo

`choco pin <pacchetto>`

Fa in modo che un pacchetto non venga aggiornato automaticamente da **choco upgrade all**

`choco pack`

Compila un pacchetto in modo da renderlo fruibile in un repository

`choco source <subcomando>`

Per visualizzare (**choco source list**) aggiungere (**choco source add**), abilitare e disabilitare i repository da cui chocolatey recupera i pacchetti

`choco list [filtro]`

Per cercare pacchetti sui repository, che in nome o descrizione contengano il filtro scelto

Il parametro <pacchetti> accetta una lista di pacchetti, oppure la keyword all che, in caso di upgrade o disinstallazione, aggiorna o disinstalla tutti i pacchetti correntemente installati.

Esistono molti **switch**; quelli più usati sono **-y** (esegue il comando senza chiedere conferma) e **--force** (lo esegue anche in caso di errore, ad esempio esegue un install anche se il pacchetto risulta già installato).

Repository interni

Normalmente il pacchetto contiene, nei repository pubblici, solo gli script di gestione che vengono richiamati dai comandi chocolatey; è dunque un metapackage.

Quando vogliamo far funzionare chocolatey all'interno di una rete locale, in alcuni condizioni è necessario un repository interno di pacchetti, ad esempio se serve:

- la distribuzione interna di software non disponibile sui repository pubblici
- l'installazione su client che non possono raggiungere la rete esterna

Nel primo caso si può effettuare il packaging degli installer all'interno dei pacchetti, abilitando anche i repository pubblici per recuperare pacchetti liberamente disponibili.
I repository pubblici applicano anche politiche di **rate limiting** per impedirne l'uso troppo intensivo da parte dei client.

In ogni caso può essere necessario costruire un repository interno alla propria rete, e saper modificare e creare pacchetti.

L'operazione di conversione di un pacchetto pubblico contenente solo script, a quella di un pacchetto ad uso locale contenente anche gli installer, è detta **internalizzazione**.

Repository interni

Negli esempi ci riferiremo a uno share SMB, la soluzione più semplice (specialmente se si ha a disposizione un Dominio Windows), utilizzabile inserendo in una unica directory condivisa tramite SMB i pacchetti nupkg.
Non offre indicizzazione ed altre feature avanzate; è consigliabile utilizzarla (secondo la documentazione ufficiale) solo fino all'hosting di 500 pacchetti.

Per un'uso più intensivo, ci sono soluzioni di diverso tipo; quelle principali sono basate su un web service; fra queste, in ambiente IIS è disponibile Nuget.Server.
Per web server diversi, è disponibile ad esempio Php Nuget (open source).
Altre soluzioni sono offerte dal progetto Chocolatey con licenza Business.

La directory che indicheremo negli esempi è:

```
\\w2ksrv6\Software\WindowsNetInstall\chocolatey
```

Ovvero quella che è attualmente in uso nei nostri sistemi.

Creazione di pacchetti ex novo

La creazione di pacchetti è gestita dal comando **choco new** e dal comando **choco pack**.

Choco new: I template

Un template è un set di script di installazione generici, che, dati i parametri forniti al comando

```
choco new <pacchetto> [<opzioni>] [<proprietà=valore>...]
```

vengono configurati, generando un pacchetto (struttura di directory e file) pronto per essere compilato, a meno dell'inclusione dei resource necessari (installer .exe o .msi, archivi .zip ...).
Il pacchetto è in genere composto da un file **.nuspec**, contenente metadati, e da una directory **tools**, contenente gli script di gestione ed i resource.
Alcune proprietà per il comando **choco new**:

- packageversion
- maintainername
- maintainerrepo
- installertype
- url (Il path dell'installer, che può essere incluso nel pacchetto o un URL)
- url64
- silentargs (Argomenti da passare all'eseguibile dell'installer; utile a lanciare l'installazione in background; variano secondo il tipo di installer)

Creazione di pacchetti ex novo

Il nome del pacchetto, usando il template di default, è automaticamente ciò che chocolatey si aspetterà di trovare nelle chiavi di disinstallazione nel registro di sistema, come Display Name; va quindi scelto con cura.
Alcune opzioni:

- --template, -t : seleziona uno specifico template (se non viene fornita, viene usato il template generico di base)
- --buildpackage : tenta di compilare il pacchetto immediatamente dopo averlo creato dal template.

I template già pronti, che coprono la maggior parte dei casi d'uso, sono installabili direttamente dal repository pubblico; In particolare si segnalano:

- msi.template per la creazione di pacchetti basati su installer .msi.
- zip.template per la creazione di pacchetti basati su un archivio zip contenente eseguibili e dati.

Al termine dell'installazione, i metadati e la directory **tools** vengono copiati in **%programdata%\chocolatey** per poter riutilizzare gli script. Gli eseguibili vengono inoltre creati **shim** (symlink che redirezionano anche dll e percorso di esecuzione), e vengono collegati nello Start Menu. Se alcuni file non vanno archiviati (ad es. se sono installer internalizzati), va creato in **tools** un file vuoto con lo stesso nome del file seguito da **.ignore**.

Creazione di pacchetti ex novo

Esempio: creazione del pacchetto OpenWebStart

Creiamo un pacchetto che installa l'applicativo OpenWebStart, di cui abbiamo l'installer (in formato exe). Creiamo lo scheletro di un pacchetto (file e directory necessari), con template standard, con il comando **choco new**:

```
choco new OpenWebStart --version 1.3.2 silentargs="'-q'"
url="'${toolsDir}\OpenWebStart_windows-  x64_1_3_2.exe'"
installertype=exe
```

Il parametro -q verrà fornito all'eseguibile per farlo installare in modo silenzioso, sapendo che tipologia di installer è (in questo caso, Install4j) e quindi quali parametri accetta .

Entriamo poi nella directory **tools**, e modifichiamo il file **chocolateyinstall.ps1**, sostituendo gli apici singoli in doppi alla riga:

```
$url = "${toolsDir}\OpenWebStart_windows-x64_1_3_2.exe" # download url,
HTTPS preferred
```

Questo abilita l'espansione delle variabili in powershell.

Creazione di pacchetti ex novo

Esempio: creazione del pacchetto OpenWebStart

Adesso modifichiamo **chocolateyuninstall.ps1**; la riga

```
silentArgs    = "/qn /norestart"
```

in origine per installer .msi, diventerà

```
silentArgs    = "-q"
```

Rendendo silenziosa anche la disinstallazione, per un installer di tipo Install4j.

La variabile **$toolsDir** viene definita in testa allo script di installazione, e fa riferimento alla cartella **tools** interna al pacchetto, dove copiamo il file dell'installer (**OpenWebStart_windows-x64_1_3_2.exe**).

Per evitare che l'installer venga archiviato e venga creato uno *shim*, creiamo nella directory **tools** il suo file **.ignore**, ovvero **OpenWebStart_windows-x64_1_3_2.exe.ignore**.

La creazione del pacchetto è conclusa, va quindi compilato ed inserito nel repository.

Creazione di pacchetti ex novo

Choco pack: compilazione del pacchetto

Nella directory del pacchetto appena creata va eseguito il comando di compilazione

```
C:\chocorepo>cd openwebstart
C:\chocorepo\openwebstart>choco pack
```

Se tutto va a buon fine, verrà creato un file di nome **openwebstart.1.3.2.nupkg** pronto per essere inserito sul repository.

Internalizzazione automatica di pacchetti pubblici

Se la licenza a disposizione è di tipo business, il client implementa l'internalizzazione automatica di pacchetti dei repository pubblici, la ricompilazione (tranne che in casi peculiari di script altamente customizzati) ed il push verso il repository locale.

In caso di un repository SMB il comando è semplicemente

```
 choco download <pacchetto> --internalize --resources-location
\\path\repository\locale
```

La licenza business è quindi interessante, ad esempio, nel caso che ci sia il bisogno di avere a disposizione un repository con molti pacchetti pubblici completamente internalizzati.

Internalizzazione manuale

Per internalizzare un pacchetto pubblico, senza licenza business, procediamo manualmente. Il procedimento è simile a quello di creazione ex novo, con alcune differenze.

E' possibile scaricare manualmente i pacchetti dal sito del repository comunitario, ma se il numero di pacchetti è grande, è più pratico:

- installarli in massa utilizzando chocolatey install
- recuperarli poi dalla cache di chocolatey, ripacchettizzarli e caricarli sul repository interno (internalizzarli).

Per prima cosa, scaricare e installare il pacchetto scelto. In questo esempio, ho deciso di internalizzare manualmente il pacchetto "firefox":

```
choco install firefox -y
```

Ricompilazione

Il pacchetto verrà poi a trovarsi in cache, nella directory:

```
%programdata%\chocolatey\lib\firefox
```

Il file .nupkg è quello che andremo a manipolare per poi inserirlo nel nostro repository; E' un file zip, scompattiamolo.

Eliminiamo le cartelle **_rels, packages**, e il file **[Content_Types].xml**. Questi vengono creati durante il **choco pack** e vanno rigenerati.

Internalizzazione manuale

Come nel caso della creazione di pacchetto ex novo, dovremo aprire lo script di installazione (**chocolateyinstall.ps1**) e modificare gli URL a cui recuperare gli installer, utilizzando la variabile **$toolsDir** per indicare la directory **tools** all'interno del pacchetto.

Gli eseguibili verranno poi inseriti nella directory tools, assieme ai file **.ignore** .

Adesso, aprire il file **.nuspec** per modificare eventuali tag (ad esempio, il numero di versione, o inserire il nome del maintainer interno).
A questo punto, invocare **choco pack** tramite un command prompt.

Modifica dei repository sui client

Proseguendo usando il pacchetto appena creato, la procedura è la seguente:

- Aggiungiamo il repository interno al nostro client, con priorità più alta rispetto al repository pubblico (che ha 0):

```
choco source add -n internal -
s"\\w2ksrv6\Software\WindowsNetInstall\chocolatey" --priority=100
```

Per testare il nuovo pacchetto, esso va installato.

- Installiamo il pacchetto in modalità verbosa per verificare che tutto sia in ordine

```
C:\>choco install Firefox -y --verbose
[...]
Copying Firefox
  from 'C:\ProgramData\chocolatey\lib\Firefox\tools/FirefoxSetup105.0.1-
64.exe'
Hashes match.
Installing Firefox...
[...]
The install of firefox was successful.
  Software installed to 'C:\Program Files\Mozilla Firefox'
```

Il software è stato quindi installato correttamente dal repository locale (notare la struttura di directory specifica del nostro script).

Successivi sviluppi

Chocolatey Automatic Package Updater Module
Chocolatey package provider for Puppet

Dubbi e domande?