Usługi, role, oprogramowanie

Konta usługowe MSA i gMSA, RemoteApps oraz instalacja oprogramowania.

- Instalacja i usuwanie oprogramowania
- Konta usługowe MSA i gMSA
- RemoteApps w Remote Desktop Services

Instalacja i usuwanie oprogramowania

□□ Poziom: średniozaawansowany **□□ Dotyczy: Windows 10/11, Windows Server 2016+**

□ Czas: ~25 min

Cztery różne mechanizmy instalacji współistnieją na Windowsie naraz — pakiety MSI, opcjonalne komponenty systemu, role serwera i menedżer pakietów `winget` — i każdy ma inne polecenie do sprawdzenia, czy się powiodła.

□□ Skróty użyte na tej stronie (8)

Skrót	Co znaczy
CMD	skrót od polecenia command, dawna nazwa wiersza poleceń Windows sprzed PowerShella
DNS	Domain Name System: książka telefoniczna internetu, zamienia nazwy na adresy IP
GUID	Globally Unique Identifier: unikalny w skali globalnej identyfikator obiektu, tu konkretnego protektora klucza BitLockera
MSI	Microsoft Installer: format paczki instalacyjnej Windows
PDF	Portable Document Format: format dokumentu, który wygląda tak samo na każdym komputerze, np. zakres akredytacji jednostki na stronie PCA
RMM	Remote Monitoring and Management: platforma do zdalnego zarządzania i monitorowania stacji roboczych
RSAT	Remote Server Administration Tools: dodatkowe narzędzia administracyjne Windows do zarządzania AD, DNS i GPO ze stacji roboczej
WWW	World Wide Web: strony i aplikacje otwierane w przeglądarce po HTTP albo HTTPS

Fiszka

□□ ZMIENIA SYSTEM

Instalacja pakietu MSI w trybie cichym: msiexec /qn

```
msiexec /i "SciezkaDoPliku.msi" /qn /norestart /l*v "SciezkaDoLogu.log"
```

Uruchamiamy z poziomu CMD bądź PowerShell z uprawnieniami administratora. `/qn` (quiet, no UI) instaluje bez jakiegokolwiek okna dialogowego — żadnego kreatora, żadnego paska postępu, nic do kliknięcia, co jest warunkiem koniecznym do uruchomienia instalacji z poziomu skryptu bez nadzoru. `/norestart` wstrzymuje automatyczny restart systemu, nawet jeśli instalator normalnie by go zażądał — restart wykonujemy świadomie, osobnym poleceniem, w oknie serwisowym, nie jako efekt uboczny instalacji. `/l*v` włącza szczegółowe (`verbose`) logowanie całego przebiegu instalacji do wskazanego pliku.

MSI = Microsoft Installer: format paczki instalacyjnej Windows · **CMD** = skrót od polecenia command, dawna nazwa wiersza poleceń Windows sprzed PowerShella · **msiexec** = Microsoft Installer execute: instalacja paczek MSI

ZMIENIA SYSTEM

Uruchomienie instalatora MSI z poziomu PowerShella z oczekiwaniem na zakończenie

```
$proces = Start-Process msiexec.exe -ArgumentList '/i', '"SciezkaDoPliku.msi"', '/qn', '/norestart', '/l*v', '"SciezkaDoLogu.log"' -Wait -PassThru
$proces.ExitCode
```

> **Pojęcie: Start-Process -Wait -PassThru.** Bez `-Wait` `Start-Process` uruchamia zewnętrzny program i natychmiast oddaje sterowanie z powrotem do skryptu, nie czekając na jego zakończenie — kolejna linia skryptu (na przykład sprawdzenie, czy plik zainstalowanej aplikacji już istnieje) wykonałaby się, zanim instalator w ogóle skończy pracę. `-Wait` wstrzymuje wykonanie skryptu do chwili zamknięcia się procesu, a `-PassThru` sprawia, że `Start-Process` zwraca obiekt procesu (z właściwością `ExitCode`) zamiast nic nie zwracać — bez `-PassThru` `$proces` pozostałoby puste, mimo że `-Wait` zadziałało poprawnie.

MSI = Microsoft Installer: format paczki instalacyjnej Windows

TYLKO ODCZYT

Sprawdzenie kodu wyjścia instalacji

```
if ($proces.ExitCode -eq 0) {
    Write-Host "Instalacja zakończona powodzeniem" -ForegroundColor Green
```

```
} elseif ($proces.ExitCode -eq 3010) {
    Write-Host "Instalacja zakończona powodzeniem, wymagany restart" -ForegroundColor
Yellow
} else {
    Write-Host "Instalacja zakończona błędem, kod: $($proces.ExitCode)" -ForegroundColor
Red
}
```

> **Pojęcie: kod wyjścia MSI.** `0` oznacza pełny sukces, `3010` oznacza sukces wymagający restartu (odłożonego dzięki `/norestart` z fiszki wyżej), każdy inny kod jest błędem konkretnego typu udokumentowanym przez Microsoft w liście kodów błędów Windows Installer — samo `/l*v` z poprzedniej fiszki jest tym, co pozwala doczytać w pliku logu, co dokładnie poszło nie tak przy kodzie innym niż `0` czy `3010`.

Przykład:

```
Instalacja zakończona powodzeniem, wymagany restart
```

MSI = Microsoft Installer: format paczki instalacyjnej Windows

TYLKO ODCZYT

Sprawdzenie dostępnych komponentów opcjonalnych systemu: Get-WindowsCapability

```
Get-WindowsCapability -Online | Where-Object Name -like "*Telnet*"
```

> **Pojęcie: komponent opcjonalny (Windows Capability).** Funkcje systemu Windows dostarczane jako osobne, doinstalowywane na żądanie pakiety — klient Telnet, klient/serwer OpenSSH, narzędzia RSAT to typowe przykłady. `-Online` oznacza sprawdzenie stanu na uruchomionym, bieżącym systemie, w odróżnieniu od operowania na obrazie offline.

Przykład:

```
Name  : Telnet.Client~~~~0.0.1.0
State : NotPresent
```

□□ **RSAT** = Remote Server Administration Tools: dodatkowe narzędzia administracyjne Windows do zarządzania AD, DNS i GPO ze stacji roboczej

□□ ZMIENIA SYSTEM

Instalacja komponentu opcjonalnego: Add-WindowsCapability

```
Add-WindowsCapability -Online -Name "Telnet.Client~~~~0.0.1.0"
```

Instalujemy dokładnie wskazany komponent po jego pełnej nazwie zwróconej przez `Get-WindowsCapability` — nazwa zawiera wersję i architekturę, więc kopiujemy ją zamiast wpisywać z pamięci.

Przykład:

```
Path          :
Online        : True
RestartNeeded : False
```

□□ TYLKO ODCZYT

Instalacja Telnet przez starsze narzędzie: pkgmgr

```
pkgmgr /iu:"TelnetClient"
```

Komendę uruchamiamy jako Administrator w CMD bądź PowerShell. `pkgmgr` jest starszym, poprzedzającym `Add-WindowsCapability` narzędziem instalacji komponentów opcjonalnych — na nowszych wersjach Windows 10/11 bywa już niedostępne albo oznaczone jako przestarzałe, `Add-WindowsCapability` jest dziś właściwym wyborem na nowo pisanym skrypcie. Zostawiamy to polecenie jako wiedzę pomocniczą do rozpoznania go w starszych, istniejących skryptach.

□□ **CMD** = skrót od polecenia command, dawna nazwa wiersza poleceń Windows sprzed PowerShella

□□ ZMIENIA SYSTEM

Instalacja klienta i serwera OpenSSH

```
Add-WindowsCapability -Online -Name OpenSSH.Client~~~~0.0.1.0
Add-WindowsCapability -Online -Name OpenSSH.Server~~~~0.0.1.0
```

Ten sam mechanizm co przy Telnecie — OpenSSH od Windows Server 2019 / Windows 10 1809 jest dostarczany jako komponent opcjonalny wbudowany w system, bez potrzeby pobierania zewnętrznego instalatora.

□□ ZMIENIA SYSTEM

Instalacja narzędzi RSAT na kliencie Windows

```
Get-WindowsCapability -Online | Where-Object Name -like "Rsat.ActiveDirectory*" | Add-WindowsCapability -Online
```

Przepuszczamy wynik wyszukania przez potok wprost do `Add-WindowsCapability` — wygodne, gdy nie chcemy przepisywać pełnej, długiej nazwy komponentu ręcznie. RSAT (Remote Server Administration Tools) na stacji roboczej daje dostęp do modułów takich jak `ActiveDirectory`, wykorzystywanych na stronie Skrypty audytowe Active Directory.

□□ **RSAT** = Remote Server Administration Tools: dodatkowe narzędzia administracyjne Windows do zarządzania AD, DNS i GPO ze stacji roboczej

□□ ZMIENIA SYSTEM

Włączenie funkcji opcjonalnej systemu klienckiego: Enable-WindowsOptionalFeature

```
Enable-WindowsOptionalFeature -Online -FeatureName "Microsoft-Hyper-V-All" -All -NoRestart
```

> **Pojęcie: funkcja opcjonalna kontra komponent (Capability).** Funkcje opcjonalne (`Windows Optional Feature`) to inny, starszy mechanizm niż komponenty opcjonalne (`Windows Capability`) — obejmuje większe, bardziej złożone podsystemy, takie jak Hyper-V, .NET Framework 3.5 czy kontener Windows Sandbox, zarządzane osobnym poleceniem i osobną listą niż `Get-WindowsCapability`. Oba mechanizmy współistnieją na tym samym systemie i trzeba wiedzieć, do którego z nich należy dana funkcja, żeby użyć właściwego polecenia.

□□ TYLKO ODCZYT

Sprawdzenie stanu funkcji opcjonalnej

```
Get-WindowsOptionalFeature -Online -FeatureName "Microsoft-Hyper-V-All"
```

Przykład:

```
FeatureName : Microsoft-Hyper-V-All
State       : Enabled
```

□□ ZMIENIA SYSTEM

Instalacja roli serwera: Install-WindowsFeature

```
Install-WindowsFeature -Name "DNS", "RSAT-DNS-Server" -IncludeManagementTools
```

> **Pojęcie: rola i funkcja serwera.** Na Windows Server rozróżniamy role (większe zestawy funkcjonalności definiujące przeznaczenie serwera — kontroler domeny, serwer DNS, serwer plików) i funkcje (mniejsze, uzupełniające komponenty) — `Install-WindowsFeature` obsługuje oba rodzaje jednym poleceniem. `-IncludeManagementTools` dokłada konsolę zarządzania i moduły PowerShella potrzebne do administrowania rolą, nie tylko samą jej działającą funkcjonalność.

Przykład:

```
Success Restart Needed Exit Code      Feature Result
------- -------------- ---------      --------------
True    No             Success        {DNS, Remote Server Administration Tools...}
```

□□ **DNS** = Domain Name System: książka telefoniczna internetu, zamienia nazwy na adresy IP

□□ TYLKO ODCZYT

Sprawdzenie zainstalowanych ról i funkcji na serwerze

```
Get-WindowsFeature | Where-Object Installed -eq $true | Select-Object Name, DisplayName
```

Przykład:

```
Name                       DisplayName
----                       -----------
DNS                        DNS Server
```

```
FS-FileServer                File and Storage Services
Web-Server                   Web Server (IIS)
```

◫◫ ZMIENIA SYSTEM

Instalacja pakietu przez winget

```
winget install --id NazwaPakietu --silent --accept-package-agreements --accept-source-agreements
```

> **Pojęcie: winget.** Menedżer pakietów wbudowany w Windows 10/11 (i doinstalowywany na Windows Server), pobierający i instalujący oprogramowanie z centralnego repozytorium Microsoftu i repozytoriów firm trzecich, jednym poleceniem, bez ręcznego pobierania instalatora ze strony producenta. `--silent` odpowiada w duchu `/qn` z `msiexec`, `--accept-package-agreements` / `--accept-source-agreements` pomijają interaktywne pytania o akceptację licencji, co jest wymogiem do uruchomienia w skrypcie bez nadzoru.

◫◫ **winget** = Windows Package Manager: menedżer pakietów do instalacji programów w Windows

◫◫ TYLKO ODCZYT

Sprawdzenie zainstalowanych pakietów: Get-Package

```
Get-Package -Name "*NazwaPakietu*" | Select-Object Name, Version, ProviderName
```

> **Pojęcie: Get-Package.** Ogólne polecenie modułu `PackageManagement`, zwracające zainstalowane oprogramowanie niezależnie od tego, jakim mechanizmem zostało zainstalowane (MSI, `winget`, `Programs and Features`) — wygodne jako jeden punkt sprawdzenia zamiast pamiętania, którym narzędziem dany pakiet trafił na maszynę.

Przykład:

```
Name              Version      ProviderName
----              -------      ------------
seven-zip.7zip    23.01        msi
```

◫◫ **MSI** = Microsoft Installer: format paczki instalacyjnej Windows

◫◫ ZMIENIA SYSTEM

Odinstalowanie aplikacji po nazwie

```
Get-Package -Name "NazwaPakietu" | Uninstall-Package
```

Odinstalowujemy przez ten sam moduł `PackageManagement`, którym sprawdzaliśmy obecność pakietu — działa dla oprogramowania zainstalowanego mechanizmami, które ten moduł rozpoznaje (głównie MSI i pakiety `winget` zarejestrowane w standardowym rejestrze odinstalowywania).

◫◫ **MSI** = Microsoft Installer: format paczki instalacyjnej Windows

◫◫ ZMIENIA SYSTEM

Odinstalowanie po kodzie produktu MSI

```
msiexec /x {12345678-90AB-CDEF-1234-567890ABCDEF} /qn /norestart /l*v
"SciezkaDoLogu.log"
```

> **Pojęcie: kod produktu (ProductCode).** Unikalny identyfikator GUID przypisany każdemu pakietowi MSI przy jego budowaniu — pozwala odinstalować dokładnie tę wersję pakietu, nawet jeśli sam plik `.msi` użyty do instalacji już nie istnieje na dysku. Kod produktu odczytujemy z rejestru (`HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall`) albo z właściwości `IdentifyingNumber` w `Get-CimInstance Win32_Product`, jeśli pakiet nie jest już nigdzie indziej widoczny.

◫◫ **MSI** = Microsoft Installer: format paczki instalacyjnej Windows · **GUID** = Globally Unique Identifier: unikalny w skali globalnej identyfikator obiektu, tu konkretnego protektora klucza BitLockera · **HKLM** = HKEY_LOCAL_MACHINE: gałąź rejestru Windows z ustawieniami całego komputera, a nie pojedynczego użytkownika · **msiexec** = Microsoft Installer execute: instalacja paczek MSI

◫◫ TYLKO ODCZYT

Odinstalowanie pakietu winget

```
winget uninstall --id NazwaPakietu --silent
```

Symetryczne do instalacji — `winget` pamięta, co sam zainstalował, i potrafi to samo cofnąć bez konieczności szukania kodu produktu ręcznie.

winget = Windows Package Manager: menedżer pakietów do instalacji programów w Windows

ZMIENIA SYSTEM

Instalacja MSI po cichu z logiem: msiexec /i ... /qn /norestart /l*v

```
msiexec /i "SciezkaDoPliku.msi" /qn /norestart /l*v "SciezkaDoLogu.log"
```

To dokładnie ta sama komenda co w pierwszej fiszce tej strony, warta osobnej karty jako gotowy wzorzec do wklejenia w zadaniu RMM: bez `REINSTALL`, bez `REINSTALLMODE`, z logiem `/l*v` włączonym od razu, nie dopiero po tym, jak coś przestanie działać. Sekcja niżej pokazuje, dlaczego dopisanie `REINSTALL=ALL` do tej samej komendy potrafi ją po cichu zepsuć.

Uważaj

- `Get-CimInstance Win32_Product` (starsze polecenie do listowania zainstalowanych pakietów MSI) wywołuje przy każdym zapytaniu ciche przeliczenie (`reconfiguration`) wszystkich zainstalowanych pakietów MSI w tle — potrafi zauważalnie spowolnić system i generować dodatkowe wpisy w dzienniku zdarzeń, `Get-Package` tego problemu nie ma.
- `pkgmgr` jest przestarzałym narzędziem — na nowo pisanych skryptach sięgamy po `Add-WindowsCapability`, zostawiając `pkgmgr` wyłącznie do zrozumienia starszego, istniejącego kodu.
- Kod wyjścia `3010` z instalacji MSI oznacza sukces wymagający restartu — traktowanie go jako błędu w automatyzacji jest częstą pomyłką, która przerywa dalszy przebieg wdrożenia, mimo że instalacja faktycznie się powiodła.

MSI = Microsoft Installer: format paczki instalacyjnej Windows · **RMM** = Remote Monitoring and Management: platforma do zdalnego zarządzania i monitorowania stacji roboczych · **msiexec** = Microsoft Installer execute: instalacja paczek MSI

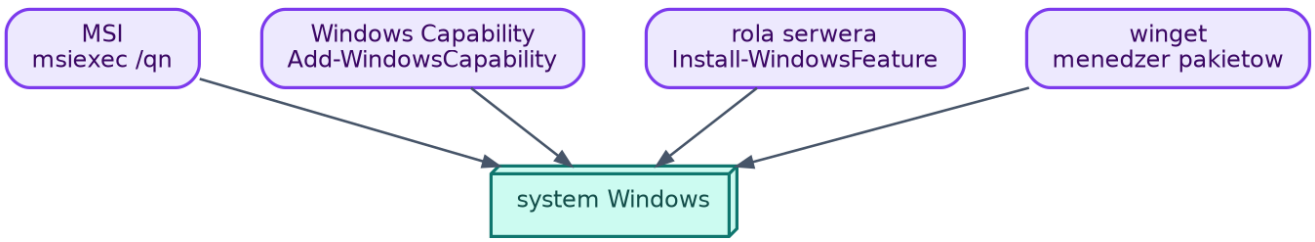
Z życia wzięte

Z ŻYCIA WZIĘTE

> Zlecamy przez RMM cichą aktualizację czytnika PDF na wszystkich stacjach w Pamiątkowie w środę wieczorem, z dopisanym na wszelki wypadek `REINSTALL=ALL`. Rano konsola RMM pokazuje same zielone haczyki i kod wyjścia `0` na każdej maszynie. Po południu okazuje się, że żadna stacja nie ma nowszej wersji, bo `REINSTALL=ALL` wyłączyło mechanizm `FindRelatedProducts`, który miał rozpoznać i usunąć starszy pakiet.

Po co to istnieje

Windows utrzymuje cztery częściowo niezależne mechanizmy dostarczania oprogramowania, bo powstawały w różnych erach systemu i obsługują różne kategorie potrzeb. MSI (Windows Installer) istnieje od Windows 2000 i pozostaje standardem dla aplikacji komercyjnych i wewnętrznych paczek firmowych — ma ustandaryzowany format, transakcyjną instalację (nieudana instalacja wycofuje wszystkie zmiany) i wbudowane wsparcie dla cichej instalacji, na czym opiera się cała pierwsza część tej strony. Komponenty i funkcje opcjonalne (`Windows Capability`, `Windows Optional Feature`) to mechanizm samego systemu operacyjnego do włączania fragmentów, które są już fizycznie na dysku (albo pobierane z Windows Update na żądanie), ale domyślnie wyłączone, żeby nie zajmować miejsca i powierzchni ataku niepotrzebnie. Role i funkcje serwera to rozszerzenie tego samego pomysłu, dostosowane do specyfiki Windows Server, gdzie „przeznaczenie" maszyny (kontroler domeny, serwer plików, serwer WWW) jest pierwszoplanową koncepcją zarządzania. `winget` jest najmłodszym z czterech — odpowiedzią na brak scentralizowanego menedżera pakietów w stylu tych znanych z Linuksa, pozwalającym zainstalować dowolną, zewnętrzną aplikację jednym poleceniem bez ręcznego szukania instalatora.



Cztery niezalezne mechanizmy dostarczania oprogramowania na Windowsie, kazdy z wlasnym poleceniem sprawdzajacym (kliknięcie otwiera pełny rozmiar)

Jak to działa

Różnica między `Get-WindowsCapability`/`Add-WindowsCapability` a `Get-WindowsOptionalFeature`/`Enable-WindowsOptionalFeature` bywa mylona, bo z zewnątrz obie robią „to samo" — włączają coś, co

leży uśpione w systemie. Rozróżnienie sprowadza się do tego, jak dana funkcjonalność jest zapakowana wewnętrznie: komponenty (`Capability`) są mniejszymi, częściej aktualizowanymi pakietami, pobieranymi na żądanie z Windows Update, jeśli nie są jeszcze obecne lokalnie — stąd Telnet czy OpenSSH, dodane relatywnie niedawno, poszły tą drogą. Funkcje opcjonalne (`Optional Feature`) są starszym mechanizmem, obejmującym zwykle większe, bardziej złożone podsystemy obecne w obrazie systemu od instalacji — stąd Hyper-V czy stary .NET Framework 3.5 wciąż korzystają z tego wzorca. Nie ma jednej reguły pozwalającej odgadnąć z góry, do którego mechanizmu należy dana funkcja — sprawdzamy oba polecenia, jeśli szukana nazwa nie pojawia się w pierwszym z nich.

Częste błędy

Traktowanie kodu wyjścia 3010 jako błędu instalacji. Skrypt wdrożeniowy, który sprawdza wyłącznie `if ($exitCode -ne 0)`, przerywa dalsze kroki mimo w pełni udanej instalacji — właściwe sprawdzenie uwzględnia explicite `0` i `3010` jako sukces, tak jak w fiszce wyżej.

Instalacja MSI bez `/norestart` w skrypcie wdrażającym wiele pakietów po kolei. Instalator z uprawnieniem do restartu systemu potrafi uruchomić restart natychmiast po swoim zakończeniu, przerywając w połowie instalację kolejnych pakietów w tej samej sekwencji skryptu.

Szukanie komponentu wyłącznie przez `Get-WindowsOptionalFeature`, gdy w rzeczywistości jest to `Windows Capability`. Polecenie zwraca pusty wynik albo błąd `The feature name is not recognized`, sugerując brak danej funkcji w systemie, podczas gdy w rzeczywistości trzeba było sprawdzić przez `Get-WindowsCapability` — oba mechanizmy trzeba sprawdzić osobno, dopóki nie wiemy z góry, do którego dana funkcja należy.

Wdrażanie MSI przez RMM

Zlecenie z RMM na „cichą aktualizację" pakietu potrafi zakończyć się sukcesem w konsoli RMM, a mimo to nie zmienić na komputerze zupełnie nic. Winny bywa jeden dopisany parametr, o którym instalator nie mówi żadnym błędem.

> Polecenie z dopisanym `REINSTALL=ALL` (często razem z `REINSTALLMODE=vomus` albo podobnym) wygląda jak bezpieczny sposób na wymuszenie ponownej instalacji, ale w rzeczywistości wyłącza logikę `FindRelatedProducts` i `RemoveExistingProducts`, którą pakiety budowane w `WiX` (`Windows Installer XML`) opierają na tabeli `Upgrade`. Bez tej logiki instalator nie rozpoznaje starszej wersji produktu jako czegoś do podmiany, `msiexec` kończy działanie kodem `0`, a na dysku zostaje dokładnie ta wersja, która była przed zleceniem.

Poprawne polecenie do „aktualizacji przez reinstalację" nie różni się niczym od zwykłej cichej instalacji nowszego pakietu:

```
msiexec /i "SciezkaDoNowszegoPliku.msi" /qn /norestart /l*v "SciezkaDoLogu.log"
```

Sam nowszy plik `.msi`, bez `REINSTALL` i bez `REINSTALLMODE`, pozwala instalatorowi samodzielnie rozpoznać starszą wersję przez `FindRelatedProducts` i usunąć ją przez `RemoveExistingProducts`, zanim zainstaluje nową. `REINSTALL=ALL` ma sens wyłącznie przy naprawie już zainstalowanej, tej samej wersji pakietu (na przykład gdy brakuje pliku po awarii dysku), nie przy podnoszeniu wersji.

Drugi, niezwiązany z `REINSTALL` kłopot z tej samej rodziny zleceń: niektóre RMM dławią się znakiem `|` w jednolinijkowym poleceniu cmd, obcinając wszystko po nim albo zgłaszając błąd składni, mimo że ta sama komenda uruchomiona ręcznie w oknie CMD działa bez zarzutu.

```
tasklist > "%TEMP%\lista-procesow.txt"
findstr /i "NazwaProcesu" "%TEMP%\lista-procesow.txt"
```

Zamiast łączyć dwie komendy potokiem `|` w jednym poleceniu zlecanym przez RMM, kierujemy wynik pierwszej komendy do pliku tymczasowego, a potem przeszukujemy ten plik osobnym poleceniem `findstr`: dwa proste kroki zamiast jednego, który RMM potrafi po drodze uszkodzić.

□□ Ćwiczenia

> Wszystko poniżej wymaga uprawnień administratora. Instalację i deinstalację realnych pakietów testujemy na maszynie testowej, nie na stacji produkcyjnej.

Ćwiczenie 1 — pełny cykl instalacji z weryfikacją kodu wyjścia

```
$proces = Start-Process msiexec.exe -ArgumentList '/i', '"C:\Sciezka\Pakiet.msi"', '/qn',
'/norestart' -Wait -PassThru
switch ($proces.ExitCode) {
    0    { "Sukces" }
    3010 { "Sukces, wymagany restart" }
    default { "Błąd, kod: $($proces.ExitCode)" }
}
```

Uruchamiamy instalację dowolnego dostępnego pakietu MSI testowego i sprawdzamy pełną obsługę trzech możliwych ścieżek wyniku.

Odpowiedź

`switch` z blokiem `default` jest tu czytelniejszy niż seria `if`/`elseif`/`else` przy więcej niż dwóch rozróżnianych wartościach — każdy kod wyjścia inny niż `0` i `3010` trafia do gałęzi domyślnej, gdzie warto dopisać dalsze rozgałęzienie na konkretne, znane kody błędów Windows Installer, jeśli skrypt ma je rozróżniać precyzyjniej niż samo „coś poszło nie tak".

Ćwiczenie 2 — porównanie dwóch mechanizmów komponentów

```
Get-WindowsCapability -Online -Name "*OpenSSH*"
Get-WindowsOptionalFeature -Online -FeatureName "*SSH*"
```

Sprawdzamy, w którym z dwóch mechanizmów faktycznie leży OpenSSH na naszym systemie.

Odpowiedź

OpenSSH pojawia się jako wynik `Get-WindowsCapability`, nie `Get-WindowsOptionalFeature` — potwierdza to, że jest zaimplementowany jako komponent (`Capability`), pobierany na żądanie, a nie jako starsza funkcja opcjonalna wbudowana od zawsze w obraz systemu. To praktyczna demonstracja rozróżnienia z sekcji „Jak to działa" wyżej.

Pytania kontrolne

1. Dlaczego `Get-Package` jest bezpieczniejszym wyborem do rutynowego sprawdzania zainstalowanego oprogramowania niż `Get-CimInstance Win32_Product`?

Odpowiedź

`Get-CimInstance Win32_Product` przy każdym wywołaniu wymusza ciche przeliczenie konfiguracji (`reconfiguration`) wszystkich zainstalowanych pakietów MSI na maszynie — udokumentowane przez Microsoft zachowanie, które potrafi zauważalnie obciążyć system i generować dodatkowe zdarzenia w dzienniku, szczególnie przy dużej liczbie zainstalowanych pakietów. `Get-Package` odczytuje te same informacje bez wywoływania tego efektu ubocznego.

2. Skrypt wdrożeniowy instaluje pięć pakietów MSI po kolei w pętli, każdy z /qn ale bez /norestart. Co może pójść nie tak?

Odpowiedź

Instalator jednego z pakietów w środku pętli może zażądać restartu systemu natychmiast po swoim zakończeniu, jeśli sam MSI ma taką flagę ustawioną — bez `/norestart` nic nie wstrzymuje tego restartu, co przerywa wykonanie skryptu w połowie sekwencji, zanim zdąży zainstalować pozostałe pakiety z listy. `/norestart` na każdym pakiecie odkłada wszystkie potencjalne restarty na jeden, świadomie zaplanowany moment po zakończeniu całej sekwencji.

Powiązane: Skrypty PowerShell: struktura i dobre praktyki · Skrypty audytowe Active Directory · WinRM: zarządzanie zdalne

Źródła: `Get-Help Start-Process` · `Get-Help Add-WindowsCapability` · dokumentacja Microsoft Learn, „Windows Installer error codes" i „winget command line reference"

Konta usługowe MSA i gMSA

Typ: procedura **Dotyczy: Active Directory, Windows Server 2012+** **Czas: ~30 min**

Efekt: usługa albo pula aplikacji IIS uwierzytelnia się kontem, którego hasła nikt nie zna i nie wpisuje w konfiguracji — Active Directory zmienia je samo, automatycznie, zanim ktokolwiek zdąży o nim zapomnieć.

Skróty użyte na tej stronie (6)

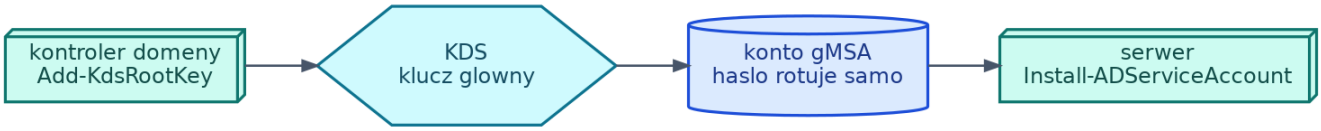
Skrót	Co znaczy
AD	Active Directory: katalog Microsoftu z kontami użytkowników, komputerów i grup w firmie
gMSA	group Managed Service Account: konto usługi, któremu Windows sam zmienia hasło
GPO	Group Policy Object: zestaw ustawień wypychanych z Active Directory na komputery
GUI	Graphical User Interface: okienka i myszka zamiast komend
IIS	Internet Information Services: serwer WWW wbudowany w Windows Server
MSA	Managed Service Account: konto usługi z automatyczną zmianą hasła, dla jednego serwera

Co realnie rozwiązuje konto usługi zarządzane

> **Pojęcie: konto usługi zarządzane (Managed Service Account).** Konto w Active Directory przeznaczone wyłącznie do uruchamiania usług i procesów, nie do logowania interaktywnego, którego hasło Active Directory zmienia samo, cyklicznie i automatycznie — bez udziału administratora i bez zapisywania go nigdzie w plikach konfiguracyjnych.

Problem, który to rozwiązuje, jest znany każdemu, kto kiedykolwiek zmieniał hasło do konta usługowego ręcznie: hasło trzeba zmienić w AD, a potem we wszystkich miejscach, gdzie jest wpisane — w konfiguracji usługi, w connection stringu, czasem w kilku aplikacjach naraz — i przy którymkolwiek z tych kroków pominiętym usługa przestaje działać, zwykle w najmniej dogodnym momencie. Konto usługi zarządzane eliminuje ten problem u źródła: hasła nigdzie nie wpisujemy, bo usługa i tak go nie potrzebuje — pobiera je automatycznie z AD w momencie startu.

Różnica MSA vs gMSA. Standardowe MSA (`Standalone Managed Service Account`) jest przypisane do jednego konkretnego komputera i nie da się go użyć na drugim. gMSA (`group Managed Service Account`) jest przypisane do grupy zabezpieczeń zawierającej komputery, którym wolno go używać — dzięki temu to samo konto usługowe obsługuje usługę rozproszoną na kilku serwerach naraz (farma IIS, klaster) bez zakładania osobnego konta na każdą maszynę. W praktyce na nowych wdrożeniach niemal zawsze wybieramy gMSA — jest funkcjonalnym nadzbiorem MSA.



Kontroler domeny generuje klucz KDS, z niego powstaje haslo konta gMSA, serwer pobiera je sam (kliknięcie otwiera pełny rozmiar)

Zanim zaczniemy

> **Z ŻYCIA WZIĘTE**
>
> Ola prosi o konto usługowe dla nowej aplikacji zespołu, rozproszonej na dwóch serwerach naraz, żeby nie musieć ręcznie synchronizować hasła między nimi po każdej wymuszonej zmianie. Zakładamy gMSA, dodajemy obie maszyny do grupy uprawnionej do pobierania hasła i instalujemy konto na każdej z nich. Od tego dnia hasło konta rotuje samo co trzydzieści dni, a żaden administrator nigdy go nie widzi ani nie wpisuje w konfiguracji.

Prace na kontrolerze domeny wymagają konta z uprawnieniami do zarządzania AD w domenie, w której chcemy użyć kont gMSA. Sprawdzamy tryb domeny i lasu — funkcja gMSA wymaga poziomu funkcjonalnego Windows Server 2012 lub wyższego:

```
(Get-ADDomain).ForestMode
(Get-ADDomain).DomainMode
```

Przebieg

1. Dodanie głównego klucza KDS

Wykonujemy raz na cały las — jeśli klucz KDS już istnieje z wcześniejszego wdrożenia gMSA, ten krok pomijamy. Ustawiamy `-EffectiveTime` na 10 godzin wstecz, żeby klucz zaczął obowiązywać natychmiast, zamiast czekać na domyślną, znacznie dłuższą propagację między kontrolerami domeny:

```
Add-KdsRootKey -EffectiveTime (Get-Date).AddHours(-10)
```

Testujemy poprawność klucza:

```
Test-KdsRootKey -KeyId (Get-KdsRootKey).KeyId
```

2. Utworzenie konta gMSA

Częstotliwość automatycznej zmiany hasła ustawiamy tylko przy tworzeniu konta — domyślnie jest to 30 dni, a zmiana tej wartości później wymaga dodatkowych kroków, więc od razu dobieramy wartość docelową:

```
New-ADServiceAccount -Name NazwaKonta -DNSHostName NazwaKonta.NazwaDomeny -
ManagedPasswordIntervalInDays LiczbaDni
```

3. Przypisanie grupy uprawnionej do pobierania hasła konta

```
Set-ADServiceAccount -Identity "NazwaKonta" -PrincipalsAllowedToRetrieveManagedPassword
"NazwaGrupy"
```

`NazwaGrupy` to grupa zabezpieczeń zawierająca konta komputerów, którym wolno korzystać z tego gMSA — nie konta użytkowników, tylko konta komputerów serwerów, na których usługa faktycznie działa.

4. Instalacja konta na serwerze docelowym

Na serwerze, który ma korzystać z konta, instalujemy przystawki PowerShell do zarządzania AD, jeśli nie są jeszcze zainstalowane:

```
Add-WindowsFeature RSAT-AD-PowerShell
```

Resetujemy bilet uwierzytelniania Kerberos, żeby serwer na pewno odświeżył swoje członkostwo w grupie z poprzedniego kroku:

```
klist purge -li 0x3e7
```

Instalujemy konto na serwerze:

```
Install-ADServiceAccount -Identity "NazwaKonta"
```

5. Przypisanie konta do usługi

Przez GUI (*Właściwości usługi* → zakładka *Logowanie* → *To konto*, wpisując `NazwaDomeny\NazwaKonta$` bez podawania hasła), albo przez PowerShell:

```
sc.exe config NazwaUslugi obj= "NazwaDomeny\NazwaKonta$" password= ""
```

Dla puli aplikacji IIS ustawiamy identyfikator tożsamości w Menedżerze IIS (*Application Pools* → *Advanced Settings* → *Identity* → *Custom account*), podając `NazwaDomeny\NazwaKonta$` również bez hasła — IIS pobiera je automatycznie z AD tak samo jak każda inna usługa systemowa.

6. Uprawnienie „Log on as a service"

Konto gMSA przypisane do usługi przez Menedżera usług albo IIS otrzymuje to uprawnienie automatycznie w większości scenariuszy. Jeśli usługa mimo poprawnej instalacji konta odmawia startu z błędem logowania, sprawdzamy wprost w polityce lokalnej (`secpol.msc` → *Local Policies* → *User Rights Assignment* → *Log on as a service*) albo w odpowiednim GPO, czy `NazwaDomeny\NazwaKonta$` rzeczywiście tam figuruje.

Weryfikacja

```
Test-ADServiceAccount -Identity NazwaKonta
```

Przykład:

```
True
```

`True` potwierdza, że serwer, na którym wykonujemy test, potrafi pobrać hasło konta z AD — jeśli usługa mimo to nie startuje, przyczyna leży już nie w samym gMSA, tylko w konfiguracji usługi albo w uprawnieniu „Log on as a service".

Dodatkowo sprawdzamy, czy usługa faktycznie działa na tym koncie:

```
Get-Service NazwaUslugi | Select-Object Name, Status, StartType
Get-CimInstance Win32_Service -Filter "Name='NazwaUslugi'" | Select-Object StartName
```

Wycofanie zmian

Odinstalowanie konta z serwera, jeśli usługę przenosimy na inną maszynę albo wracamy do zwykłego konta domenowego:

```
Uninstall-ADServiceAccount -Identity NazwaKonta
```

Usunięcie samego konta z AD, jeśli nie jest już używane przez żaden serwer:

```
Remove-ADServiceAccount -Identity NazwaKonta
```

Gdy coś nie zadziała

Objaw	Przyczyna	Naprawa
`Test-ADServiceAccount` zwraca `False`	serwer nie jest członkiem grupy z `PrincipalsAllowedToRetrieveManagedPassword`, albo członkostwo się jeszcze nie zreplikowało	sprawdzamy grupę, czekamy na replikację AD albo wymuszamy ją ręcznie, resetujemy bilet Kerberos (`klist purge`)
`Add-KdsRootKey` / `Install-ADServiceAccount` działa dopiero po kilku godzinach	domyślny czas propagacji klucza KDS w lesie	używamy `-EffectiveTime (Get-Date).AddHours(-10)` przy tworzeniu klucza, jak w kroku 1
Usługa nie startuje mimo poprawnego `Test-ADServiceAccount`	brak uprawnienia „Log on as a service" dla konta	dodajemy konto w `secpol.msc` albo w odpowiednim GPO
`Install-ADServiceAccount` zwraca błąd dostępu	brak zainstalowanego modułu `RSAT-AD-PowerShell`, albo konto wykonujące instalację nie ma uprawnień do odczytu obiektu w AD	instalujemy moduł, sprawdzamy uprawnienia delegacji w OU, gdzie leży konto gMSA

Powiązane: RemoteApps w Remote Desktop Services

Źródła: notatki własne, na podstawie „Secure group managed service accounts" — Microsoft Learn

RemoteApps w Remote Desktop Services

□□ Typ: fiszka **□□ Dotyczy: Remote Desktop Services (RDS), Windows Server 2016+**

Efekt: publikujemy, zmieniamy i usuwamy aplikacje zdalne w kolekcji RDS z poziomu PowerShella, bez klikania w konsoli za każdym razem.

□□ Skróty użyte na tej stronie (2)

Skrót	Co znaczy
NTFS	New Technology File System: system plików Windows z wbudowanymi uprawnieniami ACL na poziomie pliku i folderu
RDS	Remote Desktop Services: zdalne pulpity i aplikacje Windows Server

Fiszka

□□ ZMIENIA SYSTEM

Wylistowanie wszystkich aplikacji w kolekcji: Get-RDRemoteApp

```
# Wskazanie nazwy kolekcji z której pobieramy aplikacje
$collectionName = "NazwaKolekcji"

# Zaimportowanie modułu RemoteDesktop
Import-Module RemoteDesktop

# Pobieranie wszystkich RemoteApp z kolekcji
$remoteApps = Get-RDRemoteApp -CollectionName $collectionName

# Wyświetlanie pobranych aplikacji RemoteApp
$remoteApps | Select-Object Alias, DisplayName, FilePath
```

Przykład:

```
Alias        DisplayName            FilePath
-----        -----------            --------
notepad      Notatnik               C:\Windows\System32\notepad.exe
ksiegowosc   Aplikacja Księgowość   C:\Program Files\Ksiegowosc\ksiegowosc.exe
```

ZMIENIA SYSTEM

Dodanie nowej aplikacji do kolekcji: New-RDRemoteApp

```
# Pobieranie nazwy kolekcji do której chcemy dodać aplikację
$collectionName = "NazwaKolekcji"

# Pobieranie parametrów nowej aplikacji
$lnkFilePath = "SciezkaDoPliku\Skrot.lnk"
$iconFilePath = "SciezkaDoPliku\ikona.ico"
$appAlias = "AliasAplikacji"
$appDisplayName = "NazwaAplikacjiWyswietlanaUzytkownikom"

# Importowanie potrzebnego modułu PowerShell
Import-Module RemoteDesktop

# Dodanie nowej aplikacji
New-RDRemoteApp -CollectionName $collectionName -Alias $appAlias -DisplayName
$appDisplayName -FilePath $lnkFilePath -IconPath $iconFilePath
```

Publikujemy skrót (`.lnk`), nie bezpośrednio plik `.exe` — dzięki temu ikona i ewentualne parametry uruchomienia zapisane w skrócie trafiają razem z aplikacją do kolekcji.

ZMIENIA SYSTEM

Zmiana nazwy i parametrów aplikacji: Set-RDRemoteApp

```
# Pobieranie nazwy kolekcji i aplikacji do zmiany
$collectionName = "NazwaKolekcji"
$alias = "StaraNazwaAplikacji"
```

```
# Pobieranie parametrów do zmiany
$newDisplayName = "NowaNazwaAplikacji"
$newProgramPath = "SciezkaDoPliku\aplikacja.lnk"

# Importowanie odpowiedniego modułu PowerShell
Import-Module RemoteDesktop

# Zmiana nazwy aplikacji
Set-RDRemoteApp -CollectionName $collectionName -Alias $alias -DisplayName
$newDisplayName -FilePath $newProgramPath
```

alias = nadaj poleceniu własny, krótszy skrót na czas sesji powłoki

OSTROŻNIE · NIEODWRACALNE

Usunięcie aplikacji z kolekcji: Remove-RDRemoteApp

```
# Pobieranie nazwy kolekcji i aplikacji do usunięcia
$collectionName = "NazwaKolekcji"
$alias = "AliasAplikacjiDoUsuniecia"

# Importowanie odpowiedniego modułu PowerShell
Import-Module RemoteDesktop

# Usunięcie aplikacji z kolekcji
Remove-RDRemoteApp -CollectionName $collectionName -Alias $alias
```

alias = nadaj poleceniu własny, krótszy skrót na czas sesji powłoki

ZMIENIA SYSTEM

Przypisanie aplikacji do grupy użytkowników

```
Set-RDRemoteApp -CollectionName $collectionName -Alias $appAlias -UserGroups
"NazwaDomeny\NazwaGrupy"
```

Bez jawnego przypisania grupy aplikacja bywa domyślnie widoczna dla wszystkich użytkowników mających dostęp do kolekcji — dla aplikacji ograniczonych do jednego działu warto ograniczyć widoczność już na tym poziomie, zamiast tylko uprawnieniami NTFS na samym pliku.

□□ **NTFS** = New Technology File System: system plików Windows z wbudowanymi uprawnieniami ACL na poziomie pliku i folderu

□□ ZMIENIA SYSTEM

Parametry wiersza poleceń aplikacji

```
Set-RDRemoteApp -CollectionName $collectionName -Alias $appAlias -CommandLineSetting
Require -RequiredCommandLine "/tryb:produkcja"
```

`-CommandLineSetting` przyjmuje `Allow` (użytkownik może dopisać własne parametry), `DoNotAllow` (żadnych parametrów) albo `Require` (parametry sztywno narzucone przez administratora, jak w przykładzie wyżej) — istotne dla aplikacji, które muszą zawsze wystartować w konkretnym trybie.

Z życia wzięte

□□ Z ŻYCIA WZIĘTE

Kasia dzwoni z hotelu w środę wieczorem, bo potrzebuje pilnie wystawić fakturę klientowi, a zabrała ze sobą tylko prywatny laptop bez firmowego oprogramowania. Łączy się przez RemoteApp do aplikacji księgowej opublikowanej na rds01 i na ekranie dostaje tylko okno samej aplikacji, bez widoku na cały pulpit serwera. Faktura trafia do wydruku na jej drukarce w pokoju hotelowym, tak jakby program działał lokalnie.

Czym jest RemoteApp

Pojęcie: RemoteApp. Funkcja Remote Desktop Services, która publikuje pojedynczą aplikację zdalną tak, żeby wyglądała i działała u użytkownika jak program lokalny — własne okno, własna ikona na pasku zadań, bez widocznego pulpitu serwera dookoła. Użytkownik uruchamia jedną konkretną aplikację z serwera RDS, zamiast dostawać pełny zdalny pulpit z dostępem do wszystkiego, co na tym serwerze zainstalowane.

Różnica względem pełnego pulpitu zdalnego jest właśnie w tym zawężeniu — RemoteApp ogranicza ekspozycję serwera do jednego programu, co jest istotne zarówno z punktu widzenia bezpieczeństwa (użytkownik nie ma dostępu do reszty systemu), jak i wygody (nie trzeba tłumaczyć użytkownikom, czym jest pulpit zdalny — dla nich to po prostu kolejna aplikacja na

liście).



Klient laczy sie przez brame do hosta sesji, RemoteApp pokazuje tylko jedna aplikacje (kliknięcie otwiera pełny rozmiar)

Polecenia wykonujemy z poziomu PowerShella z zainstalowanym modułem `RemoteDesktop`, najwygodniej bezpośrednio na serwerze z zainstalowaną rolą Remote Desktop Services.

Kolekcje sesji

> **Pojęcie: kolekcja sesji.** Grupa serwerów RDS (Session Host) potraktowana jako jeden zasób do publikowania aplikacji i zarządzania użytkownikami — RemoteApp zawsze publikujemy do konkretnej kolekcji, nie do pojedynczego serwera z osobna, nawet jeśli kolekcja fizycznie składa się tylko z jednej maszyny.

Częste błędy

Publikowanie ścieżki bezpośrednio do pliku `.exe` zamiast do skrótu `.lnk` — traci się wtedy możliwość przekazania parametrów startowych zapisanych w skrócie, a ikona aplikacji bywa mniej czytelna niż ta przygotowana ręcznie w pliku `.ico`.

Zapominanie o przypisaniu grupy użytkowników po dodaniu aplikacji — nowo dodana aplikacja RemoteApp bywa od razu widoczna szerzej, niż powinna być, dopóki nie ograniczymy jej jawnie do właściwej grupy.

Powiązane: Konta usługowe MSA i gMSA · Kosz i profile użytkowników

Źródła: notatki własne, dokumentacja Microsoft (moduł `RemoteDesktop`)