Pobieranie plików z S3 przez PowerShell

Poziom: podstawy **Dotyczy: PowerShell 5.1 / 7, moduł AWS.Tools.S3**

Czas: ~20 min

Skrypty uruchamiane na Windows sięgają do S3 nie przez `aws s3`, tylko przez własny moduł PowerShella — z osobnym zestawem poleceń, ale tym samym modelem uprawnień co reszta AWS.

Komendy wykonujemy w PowerShell 5.1 (Windows PowerShell) albo PowerShell 7, z zainstalowanym modułem `AWS.Tools.S3`.

Skróty użyte na tej stronie (4)

Skrót	Co znaczy
API	Application Programming Interface: sposób, w jaki programy rozmawiają ze sobą
AWS	Amazon Web Services: chmura Amazonu
S3	Simple Storage Service: magazyn plików w chmurze AWS
SDK	Software Development Kit: zestaw narzędzi i bibliotek do budowania programów dla jakiejś platformy, tu zależny od hosta agenta budującego

Fiszka

ZMIENIA SYSTEM

Instalacja modułu AWS.Tools.S3

```
Install-Module -Name AWS.Tools.S3 -Scope CurrentUser
```

Instalujemy moduł tylko dla bieżącego użytkownika (`-Scope CurrentUser`) — nie wymaga to uprawnień administratora, w odróżnieniu od instalacji dla wszystkich użytkowników maszyny.

AWS = Amazon Web Services: chmura Amazonu

◫◫ ZMIENIA SYSTEM

Zapisanie poświadczeń jako nazwany profil: Set-AWSCredential

```
Set-AWSCredential -AccessKey WpisanyKluczDostepu -SecretKey WpisanyKluczTajny -StoreAs
NazwaProfilu
```

Zapisujemy klucze dostępowe raz, pod nazwaną etykietą, w bezpiecznie zaszyfrowanym magazynie profilu SDK dla bieżącego użytkownika Windows — każde kolejne wywołanie modułu odwołuje się już tylko do `NazwaProfilu`, nigdy do gołych kluczy wpisanych w treści skryptu.

Przykład:

```
Profile 'NazwaProfilu' was successfully saved.
```

◫◫ **SDK** = Software Development Kit: zestaw narzędzi i bibliotek do budowania programów dla jakiejś platformy, tu zależny od hosta agenta budującego

◫◫ ZMIENIA SYSTEM

Ustawienie domyślnego regionu: Set-DefaultAWSRegion

```
Set-DefaultAWSRegion -Region NazwaRegionu
```

Ustawiamy region domyślny dla bieżącej sesji PowerShella — bez tego każde polecenie modułu wymagałoby jawnego `-Region` przy każdym wywołaniu.

◫◫ TYLKO ODCZYT

Wylistowanie obiektów w kubełku: Get-S3Object

```
Get-S3Object -BucketName NazwaKubelka -ProfileName NazwaProfilu
```

Zwraca listę obiektów kubełka jako obiekty PowerShella, gotowe do dalszego filtrowania potokiem (`Where-Object`, `Sort-Object`) bez konieczności parsowania tekstu.

Przykład:

```
Key                           LastModified         Size
---                           ------------         ----
```

```
faktury/2026/styczen/f-001.pdf 2026-01-15 08:12:33   102934
faktury/2026/styczen/f-002.pdf 2026-01-16 08:05:11    98211
raport-styczen.csv             2026-01-15 09:00:02   481293
```

◫◫ TYLKO ODCZYT

Wylistowanie obiektów z filtrem po prefiksie: Get-S3Object -KeyPrefix

```
Get-S3Object -BucketName NazwaKubelka -KeyPrefix NazwaPrefiksu -ProfileName NazwaProfilu
```

Parametr `-KeyPrefix` filtruje po stronie API AWS, zanim wynik w ogóle trafi do PowerShella — szybsze i tańsze w transferze niż pobranie pełnej listy i odfiltrowanie jej lokalnie `Where-Object`, zwłaszcza przy kubełkach z dużą liczbą obiektów.

Przykład:

```
Key                            LastModified          Size
---                            ------------          ----
faktury/2026/styczen/f-001.pdf 2026-01-15 08:12:33   102934
faktury/2026/styczen/f-002.pdf 2026-01-16 08:05:11    98211
```

◫◫ **API** = Application Programming Interface: sposób, w jaki programy rozmawiają ze sobą · **AWS** = Amazon Web Services: chmura Amazonu

◫◫ TYLKO ODCZYT

Pobranie pojedynczego obiektu na dysk: Read-S3Object

```
Read-S3Object -BucketName NazwaKubelka -Key NazwaPliku -File "SciezkaDoPliku" -
ProfileName NazwaProfilu
```

> **Pojęcie:** **`Read-S3Object`**. Odpowiednik `aws s3 cp` z S3 do lokalnego dysku w module PowerShella — parametr `-Key` wskazuje pełny klucz obiektu w S3, a `-File` lokalną ścieżkę docelową, łącznie z nazwą pliku. Katalog docelowy musi istnieć wcześniej — `Read-S3Object` go za nas nie tworzy.

Przykład:

```
Directory: C:\pobrane


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        2026-01-25   9:04 AM         102934 f-001.pdf
```

S3 = Simple Storage Service: magazyn plików w chmurze AWS

TYLKO ODCZYT

Skopiowanie obiektu w obrębie S3: Copy-S3Object

```
Copy-S3Object -BucketName NazwaKubelka -Key NazwaPliku -DestinationBucket NazwaKubelka -DestinationKey "archiwum/NazwaPliku" -ProfileName NazwaProfilu
```

W odróżnieniu od `Read-S3Object`, `Copy-S3Object` nie ściąga danych przez lokalny komputer — kopiowanie odbywa się po stronie AWS, między dwoma kluczami (tym samym albo różnymi kubełkami), więc działa równie szybko dla pliku wielkości kilku kilobajtów, jak i kilkudziesięciu gigabajtów.

S3 = Simple Storage Service: magazyn plików w chmurze AWS · **AWS** = Amazon Web Services: chmura Amazonu

TYLKO ODCZYT

Pobranie wielu obiektów pętlą po liście z Get-S3Object

```
$obiekty = Get-S3Object -BucketName NazwaKubelka -KeyPrefix NazwaPrefiksu -ProfileName NazwaProfilu


foreach ($obiekt in $obiekty) {
    $nazwaLokalna = Split-Path -Leaf $obiekt.Key
    Read-S3Object -BucketName NazwaKubelka -Key $obiekt.Key -File "SciezkaDoPliku\$nazwaLokalna" -ProfileName NazwaProfilu
}
```

Najpierw pobieramy listę obiektów pasujących do prefiksu, a dopiero potem w pętli `foreach` pobieramy każdy z nich osobno — `Get-S3Object` nie ma trybu rekurencyjnego pobierania odpowiadającego `aws s3 sync`, więc wielokrotne pobranie budujemy sami tym wzorcem.

Przykład:

```
Directory: C:\pobrane


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        2026-01-25   9:06 AM         102934 f-001.pdf
-a----        2026-01-25   9:06 AM          98211 f-002.pdf
```

□□ TYLKO ODCZYT

Pobranie z obsługą błędów pojedynczego pliku

```
foreach ($obiekt in $obiekty) {
    $nazwaLokalna = Split-Path -Leaf $obiekt.Key
    try {
        Read-S3Object -BucketName NazwaKubelka -Key $obiekt.Key -File
"SciezkaDoPliku\$nazwaLokalna" -ProfileName NazwaProfilu -ErrorAction Stop
    }
    catch {
        Write-Warning "Nie udalo sie pobrac $($obiekt.Key): $($_.Exception.Message)"
    }
}
```

`-ErrorAction Stop` zamienia błąd, który domyślnie tylko wypisałby czerwony komunikat i poszedł dalej, w wyjątek przechwytywalny blokiem `try`/`catch` — bez tego przełącznika pojedynczy nieudany plik (np. brak uprawnień do konkretnego klucza) nie zatrzymałby pętli, ale też `catch` nigdy by się nie uruchomił, bo błąd nie zostałby zgłoszony jako wyjątek.

Z życia wzięte

□□ Z ŻYCIA WZIĘTE

O 10:15 automatyczny skrypt PowerShell przestaje pobierać nowe faktury z S3 na serwerze Windows, a nikt nie widzi żadnego czerwonego komunikatu w logu zadania. Tomek dodaje `-ErrorAction Stop` do wywołania `Read-S3Object` i uruchamia skrypt ręcznie, po czym od razu widzi wyjątek o wygasłym kluczu dostępu, który wcześniej po cichu przechodził dalej bez zatrzymania pętli. Po odświeżeniu poświadczeń przez `Set-AWSCredential -StoreAs` skrypt znowu pobiera wszystkie pliki, a brakujące faktury z ostatnich dni dociągają się przy

> najbliższym uruchomieniu.

Uwierzytelnianie profilem zamiast kluczy wpisanych w treści skryptu

Wszystkie powyższe fiszki zakładają wcześniejsze uruchomienie `Set-AWSCredential -StoreAs NazwaProfilu` — od tego momentu `-ProfileName NazwaProfilu` w każdej komendzie odwołuje się do zapisanego, zaszyfrowanego magazynu poświadczeń, zamiast trzymać klucz dostępu i klucz tajny jawnie w treści skryptu `.ps1`. Skrypt trafiający do repozytorium albo udostępniany innej osobie nie ujawnia wtedy żadnego sekretu — ujawniłby go dopiero skrypt z kluczami wpisanymi wprost jako zmienne. Szersze podejście do bezpiecznego trzymania haseł i sekretów w skryptach PowerShell opisuje strona Poświadczenia i bezpieczne hasła w skryptach.

Częste błędy

Wpisanie `-AccessKey` i `-SecretKey` bezpośrednio w każdym wywołaniu zamiast raz przez `-StoreAs`. Klucz tajny trafia wtedy do historii PowerShella (`Get-History`) i do samego skryptu w postaci jawnej — `Set-AWSCredential -StoreAs` istnieje właśnie po to, żeby zapisać poświadczenia raz i odwoływać się do nich nazwą.

Brak `-ErrorAction Stop` przy pobieraniu w pętli. Bez tego przełącznika błąd pojedynczego pliku (np. wygasły klucz w trakcie długiego pobierania wielu plików) przechodzi bez zatrzymania pętli i bez trafienia do `catch` — łatwo nie zauważyć, że część plików w ogóle się nie pobrała, dopóki nikt nie policzy ręcznie wyniku.

Zakładanie, że katalog docelowy `-File` zostanie utworzony automatycznie. `Read-S3Object` zwraca błąd `Could not find a part of the path`, jeśli katalog wskazany w ścieżce docelowej jeszcze nie istnieje — katalog trzeba utworzyć wcześniej, np. `New-Item -ItemType Directory -Path SciezkaDoPliku -Force`.

□□ Ćwiczenia

> Zadania wykonujemy w PowerShell z zainstalowanym modułem `AWS.Tools.S3` i dostępem do testowego kubełka.

1. Instalujemy moduł `AWS.Tools.S3`, zapisujemy poświadczenia testowego użytkownika komendą `Set-AWSCredential -StoreAs` i sprawdzamy komendą `Get-S3Object`, że profil faktycznie działa.
2. Pobieramy pojedynczy plik komendą `Read-S3Object` do katalogu, który jeszcze nie istnieje — obserwujemy dokładną treść błędu i poprawiamy go, tworząc katalog wcześniej.
3. Piszemy pętlę pobierającą wszystkie obiekty pod wybranym prefiksem, dodajemy obsługę błędów blokiem `try`/`catch` i sprawdzamy jej działanie, odbierając tymczasowo uprawnienia do jednego z plików.
4. Wyjaśniamy, dlaczego `Copy-S3Object` jest szybsze od pobrania przez `Read-S3Object` i ponownego wgrania przez odpowiednik `Write-S3Object`, mimo że efekt końcowy — kopia obiektu pod nowym kluczem — jest ten sam.

Powiązane: S3: kubełki, obiekty, uprawnienia · Konto i uprawnienia do pobierania plików z S3 · Poświadczenia i bezpieczne hasła w skryptach

Źródła: dokumentacja AWS Tools for PowerShell, moduł `AWS.Tools.S3` — polecenia `Get-S3Object`, `Read-S3Object`, `Copy-S3Object`, `Set-AWSCredential`

Revision #36
Created 2026-09-25 08:26:23 UTC by Tomasz Konieczny
Updated 2026-10-26 14:16:26 UTC by Tomasz Konieczny