Podstawy i AWS

Pojęcia chmurowe, AWS CLI oraz S3.

- Chmura: pojęcia, które trzeba znać zanim cokolwiek uruchomimy
- AWS CLI: instalacja, profile, poświadczenia
- S3: kubełki, obiekty, uprawnienia
- Konto i uprawnienia do pobierania plików z S3
- Pobieranie plików z S3 przez PowerShell
- Backup bazy do S3: agent backint i jego ograniczenia

Chmura: pojęcia, które trzeba znać zanim cokolwiek uruchomimy

Poziom: podstawy **Dotyczy: AWS, Azure — pojęcia uniwersalne** **Czas: ~20 min**

Zanim wydamy pierwszą komendę `aws` czy `az`, warto ustalić wspólny język — kto za co odpowiada, czym różni się region od strefy dostępności i dlaczego rachunek na koniec miesiąca potrafi zaskoczyć.

Skróty użyte na tej stronie (12)

Skrót	Co znaczy
AWS	Amazon Web Services: chmura Amazonu
BYOL	Bring Your Own License: przeniesienie własnej licencji na oprogramowanie do chmury zamiast licencji wliczonej w cenę
CPU	Central Processing Unit: procesor
EC2	Elastic Compute Cloud: usługa AWS wynajmująca gotowe maszyny wirtualne na godziny
IaaS	Infrastructure as a Service: chmura wynajmująca surowe maszyny i dyski
IAM	Identity and Access Management: zarządzanie tożsamościami i uprawnieniami (w AWS: użytkownicy, role, polityki)
PaaS	Platform as a Service: chmura dająca gotową platformę, np. bazę danych, bez zarządzania serwerem
RAM	Random Access Memory: pamięć operacyjna
RBAC	Role-Based Access Control: uprawnienia nadawane rolom, a role ludziom
RDS	Remote Desktop Services: zdalne pulpity i aplikacje Windows Server
S3	Simple Storage Service: magazyn plików w chmurze AWS
SaaS	Software as a Service: gotowa aplikacja w chmurze, np. poczta w przeglądarce

Z życia wzięte

Z ŻYCIA WZIĘTE

O 9:15 dział bezpieczeństwa pyta, dlaczego kubełek S3 z fakturami klientów był przez tydzień czytelny dla każdego z linkiem, skoro „przecież dane leżą u AWS, więc AWS ich pilnuje". Tomek tłumaczy, że dostawca odpowiada za bezpieczeństwo chmury jako infrastruktury, ale to zespół odpowiada za politykę dostępu do własnych danych w chmurze, więc źle skonfigurowana polityka kubełka nigdy nie jest winą dostawcy. Poprawia politykę na dostęp wyłącznie dla wskazanych kont i dopisuje sprawdzenie uprawnień kubełków do listy rzeczy przeglądanych po każdym wdrożeniu.

Model usługowy: IaaS, PaaS, SaaS i granica odpowiedzialności

Pojęcie: model usługowy. Sposób opisania, ile z warstwy technicznej (sprzęt, wirtualizacja, system operacyjny, runtime, aplikacja, dane) utrzymuje dostawca chmury, a ile zostaje po naszej stronie. Im więcej warstw bierze na siebie dostawca, tym mniej mamy kontroli, ale też mniej obowiązków utrzymaniowych.

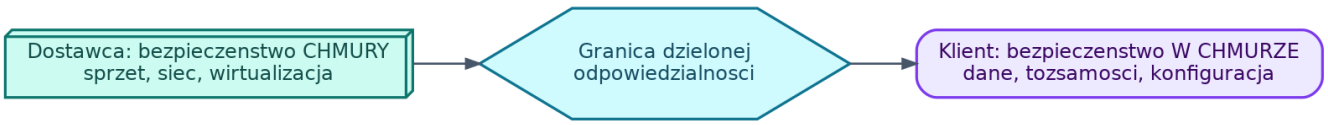
Rozróżniamy trzy podstawowe modele:

- **IaaS (Infrastructure as a Service)** — dostawca daje nam maszynę wirtualną, dysk i sieć. Sami instalujemy system operacyjny, łatamy go, konfigurujemy usługi i aplikacje. Przykład: EC2 w AWS, Virtual Machine w Azure.
- **PaaS (Platform as a Service)** — dostawca utrzymuje system operacyjny i runtime, my dostarczamy tylko kod albo dane. Przykład: RDS w AWS (silnik bazy danych bez dostępu do systemu operacyjnego pod spodem), Azure App Service.
- **SaaS (Software as a Service)** — dostawca utrzymuje całą aplikację, my jesteśmy tylko jej użytkownikiem i administratorem konfiguracji. Przykład: Microsoft 365, Salesforce.

Ten sam dostawca oferuje zwykle wszystkie trzy modele naraz dla różnych usług — wybór między EC2 (IaaS) a RDS (PaaS) do postawienia bazy danych jest wyborem tego, ile administracji silnikiem bazy chcemy sami wziąć na siebie, opisanym szerzej na stronach PostgreSQL — architektura i Instalacja PostgreSQL z własnym katalogiem danych.

Shared responsibility model — najważniejsze pojęcie na tej stronie

> **Pojęcie: shared responsibility model (model dzielonej odpowiedzialności).** Formalny podział obowiązków bezpieczeństwa i utrzymania między dostawcę chmury a klienta, publikowany przez każdego dostawcę wprost (AWS ma dokument „Shared Responsibility Model", Azure „Shared responsibility in the cloud"). Dostawca zawsze odpowiada za bezpieczeństwo *chmury* — fizyczne centra danych, sprzęt sieciowy, wirtualizator, a w usługach zarządzanych też za system operacyjny i silnik. Klient zawsze odpowiada za bezpieczeństwo *w chmurze* — konfigurację, dane, tożsamości, sieć logiczną i to, co sam na tej infrastrukturze uruchomił.



Dostawca odpowiada za bezpieczeństwo samej chmury, klient za bezpieczeństwo swoich danych i konfiguracji w chmurze (kliknięcie otwiera pełny rozmiar)

Granica tej odpowiedzialności przesuwa się razem z modelem usługowym:

Warstwa	IaaS (np. EC2, Virtual Machine)	PaaS (np. RDS, App Service)	SaaS (np. Microsoft 365)
Dane i uprawnienia dostępu	klient	klient	klient
Konfiguracja aplikacji	klient	klient	klient (w ramach opcji, które daje SaaS)
System operacyjny i jego łatanie	klient	dostawca	dostawca
Silnik/runtime	klient	dostawca	dostawca
Wirtualizacja i sprzęt	dostawca	dostawca	dostawca
Fizyczne centrum danych	dostawca	dostawca	dostawca

Rozumiemy to jako punkt wyjścia do każdej decyzji o tym, co jeszcze musimy sami zabezpieczyć — źle odczytana granica odpowiedzialności jest jedną z najczęstszych przyczyn incydentów w chmurze: administrator zakłada, że skoro dane leżą u dostawcy, to dostawca też pilnuje uprawnień dostępu do nich, a to nieprawda przy żadnym z trzech modeli.

Uważaj: przeniesienie obciążenia do chmury nie zmniejsza automatycznie naszej odpowiedzialności za dane — zmniejsza tylko zakres infrastruktury, którą musimy administrować.

Analogia

Wynajem mieszkania w bloku dobrze oddaje ten podział. Deweloper i zarządca budynku odpowiadają za fundamenty, klatkę schodową, dach i instalacje w ścianach — to jest „bezpieczeństwo budynku", odpowiednik bezpieczeństwa chmury. Nikt z lokatorów nie sprawdza samodzielnie, czy dach nie przecieka, tak jak żaden klient chmury nie sprawdza samodzielnie, czy dysk w centrum danych dostawcy jest sprawny fizycznie.

Ale to, czy zamek w drzwiach mieszkania jest zamknięty, komu dajemy zapasowy klucz i czy zostawiamy okno otwarte, wyjeżdżając na wakacje — to odpowiedzialność lokatora, nie zarządcy budynku. Zarządca nie wejdzie i nie zamknie za nas okna, nawet gdyby chciał — nie ma do tego uprawnień ani wiedzy o tym, że akurat wyjechaliśmy. Dokładnie tak samo dostawca chmury nie skonfiguruje za nas reguł dostępu do naszych danych — może dać narzędzie (zamek), ale to, czy jest właściwie użyte, zależy wyłącznie od nas.

Granica tej analogii: w mieszkaniu w wynajmowanym budynku i tak z reguły mamy pełną kontrolę nad wnętrzem. W modelu PaaS czy SaaS ta kontrola jest węższa niż nad własnym mieszkaniem — bliżej pokoju w hotelu, gdzie obsługa ma własny klucz i wchodzi sprzątać według swoich zasad. Im wyżej w modelu usługowym, tym mniej to „nasze cztery ściany", a bardziej przestrzeń współdzielona na warunkach dostawcy.

Region i strefa dostępności

> **Pojęcie: region.** Geograficznie wydzielony obszar, w którym dostawca chmury utrzymuje własną, niezależną infrastrukturę — kilka fizycznie oddzielonych centrów danych połączonych szybkimi łączami. Region w Europie Środkowej to na przykład `eu-central-1` we Frankfurcie (AWS) albo `polandcentral` w Warszawie (Azure). Wybór regionu wpływa na opóźnienia sieciowe do użytkowników, na to, jakie usługi w ogóle są w nim dostępne, oraz — w niektórych branżach — na wymogi prawne dotyczące miejsca przechowywania danych.

> **Pojęcie: strefa dostępności (Availability Zone).** Jedno z kilku fizycznie odseparowanych centrów danych wewnątrz tego samego regionu — z własnym zasilaniem, chłodzeniem i łączem do internetu, ale połączone z pozostałymi strefami regionu łączem o tak niskim opóźnieniu, że aplikacja może rozkładać obciążenie między strefami bez odczuwalnej straty wydajności. Awaria jednej strefy (pożar, awaria zasilania, błąd operacyjny dostawcy) nie powinna wpływać na pozostałe strefy tego samego regionu. Region typowo składa się z trzech stref dostępności, rzadziej z dwóch.

Rozkładanie zasobów między strefami dostępności tego samego regionu jest podstawowym, tanim sposobem budowania odporności na awarię — w odróżnieniu od replikacji między regionami, która kosztuje więcej (transfer między regionami jest płatny) i wnosi większe opóźnienie.

Konto, subskrypcja, projekt — nazewnictwo w AWS i Azure

Oba modele różnią się nazewnictwem jednostek organizacyjnych, mimo że rozwiązują ten sam problem: jak pogrupować zasoby i rozliczenia.

Poziom hierarchii	AWS	Azure
Jednostka rozliczeniowa najwyższego poziomu	konto AWS (Account)	subskrypcja (Subscription)
Grupowanie wielu kont/subskrypcji jednej organizacji	AWS Organizations	grupa zarządzania (Management Group)
Kontener na zasoby wewnątrz jednej jednostki rozliczeniowej	brak wydzielonego poziomu — rolę pełnią tagi i konwencja nazw	grupa zasobów (Resource Group)
Tożsamość i uprawnienia	IAM (Identity and Access Management)	Microsoft Entra ID (dawniej Azure AD)

Różnica, która najbardziej zaskakuje osoby przechodzące z jednego dostawcy do drugiego, to brak w AWS bezpośredniego odpowiednika grupy zasobów — w AWS każdy zasób istnieje samodzielnie w koncie i regionie, a jedynym sposobem pogrupowania powiązanych ze sobą zasobów (np. wszystkich elementów jednej aplikacji) są tagi. W Azure grupa zasobów jest natomiast jednostką organizacyjną pierwszej klasy — ma własne uprawnienia, można ją usunąć razem z całą zawartością jedną komendą, i praktycznie każdy zasób w Azure musi do jakiejś grupy zasobów należeć.

Tagi — po co je nadajemy

> **Pojęcie: tag.** Para klucz-wartość dopięta do zasobu chmurowego, niewpływająca na jego działanie, ale wpływająca na to, jak zasób widzimy w rozliczeniach, raportach i automatyzacji. Tag `srodowisko=produkcja` na instancji EC2 albo `Environment=Production` na maszynie wirtualnej w Azure pozwala później odfiltrować dokładnie te zasoby, które nas interesują, bez polegania na tym, że ktoś konsekwentnie nazywał je zgodnie z konwencją w samej nazwie.

Nadajemy tagi z trzech głównych powodów:

- **Rozliczenia** — podział kosztów między zespoły, projekty albo klientów w rachunkach wielotenantowych; bez tagów rachunek pokazuje sumę, z tagami da się go rozbić raportem kosztów po kluczu tagu.
- **Automatyzacja** — skrypty i narzędzia (np. wygaszanie środowisk testowych na noc) działają na zasobach wybranych po tagu, a nie po ręcznie utrzymywanej liście identyfikatorów.
- **Porządek operacyjny** — odpowiedź na pytania „czyj to zasób", „do jakiego projektu należy" i „czy wolno go usunąć" bez konieczności pytania nikogo.

Typowy zestaw tagów w organizacji korzystającej z chmury bywa udokumentowany razem z pełnym wykazem posiadanych zasobów — przy większych migracjach do chmury spotykana jest tabela z kolumnami takimi jak nazwa zasobu, jego funkcja, zainstalowane oprogramowanie i przydzielone zasoby (CPU, RAM, dysk), prowadzona jeszcze przed przeniesieniem czegokolwiek — tagi w docelowej chmurze są potem prostym przeniesieniem tych samych kolumn na właściwości zasobu.

Uważaj: tag nie jest mechanizmem bezpieczeństwa — nie ogranicza dostępu do zasobu, tylko go opisuje. Ograniczenie dostępu wymaga polityki IAM albo RBAC, niezależnie od tego, jak zasób jest otagowany.

Model kosztowy: co właściwie płacimy

Rachunek w chmurze publicznej rozbija się w praktyce na trzy składowe, które warto rozróżniać już na starcie, bo każda rośnie z innego powodu:

- **Obliczenia (compute)** — czas działania maszyny wirtualnej, kontenera albo funkcji bezserwerowej, zwykle rozliczany za godzinę albo za sekundę działania. To jedyna składowa, którą łatwo przewidzieć z góry, bo zależy od tego, ile maszyn i jak długo trzymamy uruchomionych.
- **Magazyn (storage)** — ilość faktycznie zajętych danych na dysku, w kubełku obiektowym czy w migawce, rozliczana zwykle za gigabajt-miesiąc. Rośnie powoli, ale bez limitu, jeśli nikt nie sprząta starych migawek i nieużywanych dysków.
- **Transfer wychodzący (egress)** — dane wysyłane z infrastruktury dostawcy na zewnątrz, do internetu albo do innego regionu tego samego dostawcy.

> **Pojęcie: transfer wychodzący (egress).** Ruch sieciowy opuszczający infrastrukturę dostawcy — w przeciwieństwie do ruchu wchodzącego (ingress), który u większości dostawców jest darmowy. Transfer wychodzący do internetu jest płatny za każdy przesłany

> gigabajt, a próg darmowy (jeśli w ogóle istnieje) jest zwykle niewielki.

Transfer wychodzący zaskakuje najczęściej z trzech powodów naraz: po pierwsze, jest niewidoczny na etapie projektowania — nikt nie planuje z góry, ile danych aplikacja odda na zewnątrz, bo w środowisku on-premise ten koszt nie istniał osobno, tylko był wliczony w opłacony z góry łącz internetowy. Po drugie, rośnie nieliniowo wraz z ruchem użytkowników — aplikacja, która nagle zyskuje popularność, generuje rachunek za transfer proporcjonalny do tej popularności, bez wcześniejszego ostrzeżenia. Po trzecie, sam ruch *między* strefami dostępności albo *między* regionami tego samego dostawcy też bywa płatny, mimo że nigdy nie opuszcza sieci dostawcy — architektura rozkładająca komponenty aplikacji między strefami dla odporności na awarię generuje więc koszt transferu, którego nie było przy jednej strefie.

Uważaj: kalkulator kosztów dostawcy (AWS Pricing Calculator, Azure Pricing Calculator) domyślnie zaniża prognozę, jeśli nie wpiszemy jawnie szacowanego wolumenu transferu wychodzącego — pole to nie jest wymagane do otrzymania wyceny, więc łatwo je pominąć.

Licencjonowanie oprogramowania w chmurze

Migracja własnego oprogramowania do chmury nie zwalnia z pytania o model licencjonowania — silnik bazy danych czy system operacyjny mogą być licencjonowane na procesor, na rdzeń, na użytkownika albo na urządzenie, dokładnie tak samo jak w środowisku on-premise, i ten model trzeba przenieść albo świadomie zmienić. Dostawcy chmury oferują dwie ścieżki: licencję wliczoną w cenę instancji (płacimy więcej za godzinę, ale nie musimy mieć własnej licencji) albo model przeniesienia własnej licencji, w AWS nazywany BYOL (Bring Your Own License), w Azure Hybrid Benefit. Wybór między nimi jest czystym rachunkiem — opłaca się przenieść własną licencję tylko wtedy, gdy faktycznie mamy jej niewykorzystaną pulę.

Częste błędy

Traktowanie modelu usługowego jako gwarancji bezpieczeństwa. Wybór PaaS zamiast IaaS zdejmuje z nas administrację systemem operacyjnym, ale nie zdejmuje odpowiedzialności za uprawnienia dostępu do danych — błędnie skonfigurowana reguła sieciowa albo zbyt szerokie uprawnienia użytkownika szkodzą tak samo przy RDS, jak przy własnej instalacji silnika na EC2.

Ignorowanie kosztu transferu wychodzącego przy projektowaniu architektury wielostrefowej albo wieloregionowej. Rozkład obciążenia między strefami czy regionami poprawia odporność na awarię, ale ma swoją cenę w rachunku — decyzję o architekturze warto podejmować ze świadomością obu stron tego kompromisu.

Mylenie tagu z mechanizmem uprawnień. Otagowanie zasobu jako `poufne=tak` niczego nie chroni samo z siebie — dopiero polityka odwołująca się do tego tagu (np. w IAM) może na jego podstawie coś zablokować albo dopuścić.

Ćwiczenia

> Zadania sprawdzające, bez dostępu do konta chmurowego — wystarczy kartka i długopis.

1. Opisujemy, co się zmienia w podziale odpowiedzialności między dostawcą a klientem, gdy przechodzimy z instancji EC2 z własną instalacją PostgreSQL na usługę RDS PostgreSQL.
2. Wymieniamy trzy powody, dla których zespół nadaje tagi zasobom chmurowym, i dla każdego podajemy jeden konkretny przykład raportu albo automatyzacji, którą ten tag umożliwia.
3. Aplikacja rozłożona na trzy strefy dostępności jednego regionu wymienia między nimi dane synchronizacyjne. Wyjaśniamy, dlaczego to rozwiązanie może generować koszt transferu, mimo że ruch nigdy nie opuszcza infrastruktury dostawcy.
4. Firma rozważa migrację własnego serwera z licencją Windows Server per-core do chmury. Opisujemy różnicę między wykupieniem instancji z licencją wliczoną a przeniesieniem własnej licencji (BYOL / Hybrid Benefit) i od czego zależy, która opcja jest tańsza.

Powiązane: AWS CLI: instalacja, profile, poświadczenia · Azure: podstawy operacyjne · S3: kubełki, obiekty, uprawnienia

Źródła: dokumentacja AWS „Shared Responsibility Model" · dokumentacja Microsoft „Shared responsibility in the cloud" · dokumentacja AWS Organizations · dokumentacja Azure Resource Manager

AWS CLI: instalacja, profile, poświadczenia

Poziom: podstawy **Dotyczy: AWS CLI v2** **Czas: ~25 min**

Zanim wykonamy pierwszą komendę `aws`, narzędzie musi wiedzieć, kim jesteśmy — profile i poświadczenia opisane niżej odpowiadają na to pytanie przy każdym kolejnym wywołaniu.

Komendy wykonujemy z CLI bash (na Windows analogicznie z PowerShella) z zainstalowanym pakietem `awscli` w wersji 2.

Skróty użyte na tej stronie (12)

Skrót	Co znaczy
API	Application Programming Interface: sposób, w jaki programy rozmawiają ze sobą
ARN	Amazon Resource Name: jednoznaczna nazwa zasobu AWS, na przykład arn:aws:s3:::nazwa-kubelka
ASCII	American Standard Code for Information Interchange: podstawowy zestaw znaków (litery, cyfry) używany w komputerach
AWS	Amazon Web Services: chmura Amazonu
CI	Continuous Integration: automatyczne budowanie i testowanie kodu po każdej zmianie; w ITIL także configuration item, czyli element konfiguracji: każdy składnik usługi zapisany w CMDB
CLI	Command Line Interface: praca przez wpisywanie komend w terminalu
EC2	Elastic Compute Cloud: usługa AWS wynajmująca gotowe maszyny wirtualne na godziny
ECS	Elastic Container Service: usługa AWS do uruchamiania kontenerów, na własnych instancjach albo na Fargate
IAM	Identity and Access Management: zarządzanie tożsamościami i uprawnieniami (w AWS: użytkownicy, role, polityki)
JSON	JavaScript Object Notation: tekstowy format danych z klamrami i cudzysłowami
MFA	Multi-Factor Authentication: logowanie drugim składnikiem, np. kodem z telefonu
S3	Simple Storage Service: magazyn plików w chmurze AWS

Fiszka

□□ TYLKO ODCZYT

Instalacja AWS CLI w wersji 2 na Linuksie

```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
```

Pobieramy oficjalny instalator w postaci archiwum ZIP — AWS CLI v2 nie jest dystrybuowane jako pakiet w standardowych repozytoriach większości dystrybucji, więc instalujemy je ręcznie z archiwum dostawcy.

```
unzip awscliv2.zip
```

Rozpakowujemy pobrane archiwum do katalogu roboczego.

```
sudo ./aws/install
```

Instalujemy binarkę `aws` do `/usr/local/bin`, dostępną potem globalnie w zmiennej `PATH`.

□□ **AWS** = Amazon Web Services: chmura Amazonu · **CLI** = Command Line Interface: praca przez wpisywanie komend w terminalu · **curl** = client URL: pobieranie i wysyłanie przez HTTP z terminala

□□ TYLKO ODCZYT

Weryfikacja wersji

```
aws --version
```

Sprawdzamy, czy instalacja się powiodła i jaka dokładnie wersja jest zainstalowana — przydatne przy diagnozowaniu różnic w zachowaniu komend między v1 a v2.

Przykład:

```
aws-cli/2.15.30 Python/3.11.8 Linux/5.14.0-427.el9.x86_64 exe/x86_64.rhel.9
```

□□ **aws** = AWS CLI: narzędzie chmury Amazonu

□□ TYLKO ODCZYT

Skonfigurowanie domyślnego profilu: aws configure

```
aws configure
```

Uruchamiamy interaktywny kreator, który zapisuje klucz dostępu, klucz tajny, domyślny region i domyślny format wyjścia do plików `~/.aws/credentials` i `~/.aws/config` — bez podania nazwy profilu trafiają one do profilu `default`, używanego wtedy, gdy żaden inny profil nie został jawnie wskazany.

Przykład:

```
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [None]: eu-central-1
Default output format [None]: json
```

aws = AWS CLI: narzędzie chmury Amazonu

TYLKO ODCZYT

Skonfigurowanie dodatkowego profilu: aws configure --profile

```
aws configure --profile NazwaProfilu
```

Ten sam kreator, ale zapisujący dane pod nazwaną sekcją zamiast `default` — pozwala trzymać na jednej maszynie poświadczenia do wielu kont AWS albo do kont różnych klientów naraz, bez nadpisywania jednego zestawu drugim.

Przykład:

```
AWS Access Key ID [None]: AKIAJZXK4EXAMPLEKEY
AWS Secret Access Key [None]: 7MDENG/bPxRfiCYEXAMPLEKEYwJalrXUtnFEMI/K
Default region name [None]: eu-west-1
Default output format [None]: table
```

AWS = Amazon Web Services: chmura Amazonu · **aws** = AWS CLI: narzędzie chmury Amazonu

TYLKO ODCZYT

Wskazanie profilu przy pojedynczym wywołaniu

```
aws s3 ls --profile NazwaProfilu
```

Dopisujemy `--profile` do dowolnej komendy, żeby jednorazowo użyć innego zestawu poświadczeń niż `default`, bez zmiany zmiennych środowiskowych ani plików konfiguracyjnych.

aws = AWS CLI: narzędzie chmury Amazonu

ZMIENIA SYSTEM

Wskazanie profilu na cały czas trwania sesji powłoki

```
export AWS_PROFILE=NazwaProfilu
```

Ustawiamy zmienną środowiskową, którą AWS CLI odczytuje automatycznie przy każdym kolejnym wywołaniu w tej samej sesji powłoki — wygodniejsze niż dopisywanie `--profile` do każdej komendy, gdy pracujemy dłużej na jednym koncie.

AWS = Amazon Web Services: chmura Amazonu · **CLI** = Command Line Interface: praca przez wpisywanie komend w terminalu · **export** = ustaw zmienną środowiskową widoczną dla uruchamianych programów

ZMIENIA SYSTEM

Wskazanie poświadczeń bez pliku konfiguracyjnego

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
export AWS_DEFAULT_REGION=eu-central-1
```

Zmienne środowiskowe mają wyższy priorytet niż plik `~/.aws/credentials` i przydają się głównie w potokach CI/CD, gdzie sekrety wstrzykuje system automatyzacji, a nie plik na dysku agenta.

> **Pojęcie: kolejność odczytu poświadczeń.** AWS CLI szuka poświadczeń w ustalonej kolejności i zatrzymuje się na pierwszym znalezionym źródle: najpierw flagi wywołania (`--profile` razem z jego zawartością), potem zmienne środowiskowe, potem plik `~/.aws/credentials`, na końcu — jeśli proces działa na EC2, ECS czy Lambda — rola przypisana do zasobu obliczeniowego. Zmienna `AWS_PROFILE` wskazuje, którą sekcję pliku konfiguracyjnego czytać, więc jeśli jest ustawiona razem z `AWS_ACCESS_KEY_ID`,

wygrywają bezpośrednio podane zmienne kluczy, nie profil.

CI = Continuous Integration: automatyczne budowanie i testowanie kodu po każdej zmianie; w ITIL także configuration item, czyli element konfiguracji: każdy składnik usługi zapisany w CMDB · **AWS** = Amazon Web Services: chmura Amazonu · **CLI** = Command Line Interface: praca przez wpisywanie komend w terminalu · **EC2** = Elastic Compute Cloud: usługa AWS wynajmująca gotowe maszyny wirtualne na godziny · **ECS** = Elastic Container Service: usługa AWS do uruchamiania kontenerów, na własnych instancjach albo na Fargate · **export** = ustaw zmienną środowiskową widoczną dla uruchamianych programów

Z życia wzięte

Z ŻYCIA WZIĘTE

O 13:45 skrypt CI/CD Tomka nagle zaczyna kończyć się błędem `AccessDenied` przy wgrywaniu artefaktów do S3, chociaż działał bez zmian od miesięcy. Sprawdza `aws sts get-caller-identity` na agencie budującym i widzi, że tymczasowe poświadczenia z przyjętej roli wygasły w trakcie długiego builda, bo nikt nie odświeżał ich po godzinie ważności. Dodaje krok odświeżający `assume-role` przed etapem wgrywania i błąd znika przy kolejnym uruchomieniu.

Pliki konfiguracyjne

Pojęcie: `~/.aws/credentials`. Plik przechowujący klucze dostępowe, podzielony na sekcje odpowiadające profilom:

```
[default]
aws_access_key_id = AKIAIOSFODNN7EXAMPLE
aws_secret_access_key = wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY

[NazwaProfilu]
aws_access_key_id = AKIAJZXK4EXAMPLEKEY
aws_secret_access_key = 7MDENG/bPxRfiCYEXAMPLEKEYwJalrXUtnFEMI/K
```

Pojęcie: `~/.aws/config`. Plik przechowujący ustawienia niebędące sekretem — region, format wyjścia, konfigurację roli do przyjęcia. Sekcje inne niż `default` mają tu przedrostek

`profile`:

```
[default]
region = eu-central-1
output = json

[profile NazwaProfilu]
region = eu-west-1
output = table
```

Rozdzielenie tych dwóch plików ma znaczenie praktyczne — `~/.aws/config` można bezpiecznie trzymać w repozytorium z ustawieniami zespołu, `~/.aws/credentials` nigdy.

Sprawdzenie tożsamości zalogowanego profilu: aws sts get-caller-identity

```
aws sts get-caller-identity
```

Pierwsza komenda, którą wykonujemy przy każdym „nie wiem już, czym jestem zalogowany" — zwraca identyfikator konta, ARN i identyfikator użytkownika albo roli, z której aktualnie korzystamy, niezależnie od tego, czy poświadczenia pochodzą z pliku, zmiennych środowiskowych czy przyjętej roli.

Przykład:

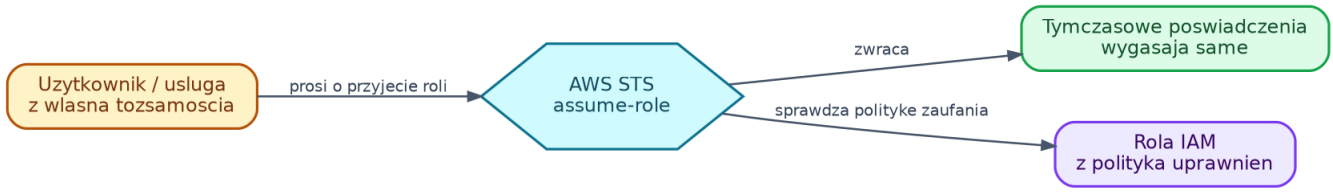
```
{
    "UserId": "AIDACKCEVSQ6C2EXAMPLE",
    "Account": "123456789012",
    "Arn": "arn:aws:iam::123456789012:user/nazwa_usera"
}
```

Ten sam wywołanie po przyjęciu roli pokazuje ARN roli zamiast ARN użytkownika:

```
{
    "UserId": "AROACKCEVSQ6C2EXAMPLE:sesja-terraform",
    "Account": "123456789012",
    "Arn": "arn:aws:sts::123456789012:assumed-role/RolaDoOdczytuS3/sesja-terraform"
}
```

Role i przyjmowanie roli

> **Pojęcie: rola IAM.** Zestaw uprawnień, który — w odróżnieniu od uprawnień przypisanych bezpośrednio użytkownikowi — nie jest na stałe związany z jedną tożsamością, tylko jest tymczasowo *przyjmowany* przez użytkownika, usługę albo konto z innej organizacji. Rola ma politykę zaufania (kto może ją przyjąć) i politykę uprawnień (co wolno robić po jej przyjęciu). Praca przez role zamiast przez długożyjące klucze dostępu jest zalecaną przez AWS praktyką — poświadczenia wynikające z przyjęcia roli są tymczasowe i wygasają same, bez potrzeby ręcznej rotacji.



assume-role sprawdza politykę zaufania roli i zwraca tymczasowe poświadczenia, które wygasają same (kliknięcie otwiera pełny rozmiar)

Przyjęcie roli: aws sts assume-role

```
aws sts assume-role --role-arn arn:aws:iam::123456789012:role/RolaDoOdczytuS3 --role-session-name sesja-terraform
```

Zwraca tymczasowy zestaw trzech wartości (klucz dostępu, klucz tajny, token sesji) ważny domyślnie przez godzinę, który trzeba potem samodzielnie wstawić do zmiennych środowiskowych albo do profilu — `assume-role` samo z siebie niczego nie konfiguruje na trwałe.

Przykład:

```
{
    "Credentials": {
        "AccessKeyId": "ASIAJEXAMPLEKEY7Q3X",
        "SecretAccessKey": "9drTJvcXLB89EXAMPLEKEYzL8vBn2u1EXAMPLE",
        "SessionToken": "FwoGZXIvYXdzEJv...EXAMPLE...TOKEN==",
        "Expiration": "2026-09-25T15:04:32+00:00"
    },
```

```
    "AssumedRoleUser": {
        "AssumedRoleId": "AROACKCEVSQ6C2EXAMPLE:sesja-terraform",
        "Arn": "arn:aws:sts::123456789012:assumed-role/RolaDoOdczytuS3/sesja-terraform"
    }
}
```

Skonfigurowanie profilu, który przyjmuje rolę automatycznie

```
[profile RolaDoOdczytuS3]
role_arn = arn:aws:iam::123456789012:role/RolaDoOdczytuS3
source_profile = default
```

Dopisujemy tę sekcję do `~/.aws/config` — od tej pory `aws s3 ls --profile RolaDoOdczytuS3` sam wykonuje `assume-role` w tle, korzystając z poświadczeń profilu `default` jako punktu wyjścia, bez ręcznego kopiowania tymczasowych kluczy.

Przyjęcie roli z wymaganym kodem MFA

```
aws sts assume-role --role-arn arn:aws:iam::123456789012:role/RolaDoOdczytuS3 --role-session-name sesja-mfa --serial-number arn:aws:iam::123456789012:mfa/nazwa_usera --token-code 123456
```

> **Pojęcie: MFA (Multi-Factor Authentication).** Drugi składnik uwierzytelnienia obok hasła albo kluczy dostępowych — zwykle sześciocyfrowy kod z aplikacji albo sprzętowego tokenu, ważny tylko przez krótki czas. Polityka zaufania roli może wymagać, żeby przyjęcie roli nastąpiło z aktywnym MFA (warunek `aws:MultiFactorAuthPresent` w polityce) — wtedy `assume-role` bez `--serial-number` i `--token-code` kończy się odmową dostępu, nawet jeśli reszta uprawnień się zgadza.

Format wyjścia i filtrowanie wyniku

Ograniczenie wyniku zapytaniem JMESPath: --query

```
aws ec2 describe-instances --query "Reservations[].Instances[].[InstanceId,State.Name]"
```

> **Pojęcie: JMESPath.** Język zapytań do struktur JSON, wbudowany w AWS CLI — pozwala wyciągnąć z rozbudowanej odpowiedzi API tylko interesujące nas pola, zamiast przepuszczać cały wynik przez `jq` w osobnym procesie. Flaga `--query` przyjmuje wyrażenie JMESPath i filtruje nim odpowiedź jeszcze przed jej wyświetleniem.

Przykład:

```
[
    [
        "i-0abcdef1234567890",
        "running"
    ],
    [
        "i-0123456789abcdef0",
        "stopped"
    ]
]
```

Filtrowanie po warunku w JMESPath

```
aws ec2 describe-instances --query
"Reservations[].Instances[?State.Name=='running'].InstanceId[]"
```

Wyrażenie w nawiasach kwadratowych z znakiem zapytania to filtr JMESPath — zwraca tylko te elementy, które spełniają podany warunek, tutaj: tylko identyfikatory instancji w stanie `running`.

Przykład:

```
[
    "i-0abcdef1234567890"
]
```

Zmiana formatu wyjścia na tabelę: --output table

```
aws ec2 describe-instances --query
"Reservations[].Instances[].[InstanceId,InstanceType,State.Name]" --output table
```

Ten sam wynik co w formacie JSON, ale sformatowany jako czytelna tabela ASCII — wygodniejszy do szybkiego przeglądu w terminalu, mniej wygodny do dalszego przetwarzania skryptem.

Przykład:

```
-------------------------------------------------
|               DescribeInstances                |
+---------------------+----------------+-----------+
|  i-0abcdef1234567890|  t3.micro      |  running   |
|  i-0123456789abcdef0|  t3.medium     |  stopped   |
+---------------------+----------------+-----------+
```

Zbiorcze zestawienie formatów wyjścia

Wartość `--output`	**Zastosowanie**
`json`	domyślny, najlepszy do dalszego przetwarzania skryptem
`yaml`	czytelniejszy dla człowieka niż `json` przy zagnieżdżonych strukturach
`text`	wartości oddzielone tabulatorami, wygodne do `awk`/`cut`
`table`	tabela ASCII, najlepsza do szybkiego przeglądu w terminalu

Częste błędy

Wpisanie kluczy dostępowych bezpośrednio w komendzie widocznej w historii powłoki. Klucz tajny trafia wtedy do `~/.bash_history` w postaci jawnej — poprawnie jest użyć `aws configure` albo zmiennych środowiskowych ustawionych w sesji, nigdy wpisywać `--secret-key` wprost, gdzie taka flaga w ogóle istnieje w innych narzędziach AWS.

Założenie, że `AWS_PROFILE` i profil w `~/.aws/config` to dwa niezależne mechanizmy. To ten sam mechanizm — zmienna środowiskowa tylko wskazuje, którą sekcję pliku odczytać. Ustawienie `AWS_PROFILE=NazwaProfilu` bez istniejącej sekcji `[profile NazwaProfilu]` w `~/.aws/config` (albo `[NazwaProfilu]` w `~/.aws/credentials`) kończy się błędem `The config profile could not be found`.

Mylenie regionu ustawionego w profilu z regionem, w którym istnieje zasób. `aws ec2 describe-instances` bez błędu zwraca pustą listę, jeśli region profilu różni się od regionu, w którym

faktycznie postawiono instancje — CLI nie ostrzega, że „szukamy w złym miejscu", tylko milczy.

□□ Ćwiczenia

> Zadania wykonujemy z zainstalowanym AWS CLI i dostępem do dowolnego konta testowego — jeśli nie mamy konta, punkty 1 i 4 rozwiązujemy opisowo.

1. Konfigurujemy dwa profile — `default` i `NazwaProfilu` — i sprawdzamy komendą `aws sts get-caller-identity --profile NazwaProfilu`, że drugi profil rzeczywiście używa innych poświadczeń.
2. Wyjaśniamy, w jakiej kolejności AWS CLI szuka poświadczeń, gdy jednocześnie ustawiona jest zmienna `AWS_PROFILE` i istnieje plik `~/.aws/credentials`.
3. Piszemy wyrażenie `--query` w JMESPath, które z wyniku `aws ec2 describe-instances` zwróci tylko identyfikatory instancji będących w stanie innym niż `running`.
4. Opisujemy różnicę między nadaniem uprawnień bezpośrednio użytkownikowi IAM a nadaniem ich przez rolę przyjmowaną komendą `assume-role` — kiedy wybieramy jedno, a kiedy drugie.

Powiązane: Chmura: pojęcia, które trzeba znać zanim cokolwiek uruchomimy · S3: kubełki, obiekty, uprawnienia · Konto i uprawnienia do pobierania plików z S3 · Terraform: podstawy

Źródła: dokumentacja AWS CLI v2 · dokumentacja AWS IAM (role, `assume-role`) · specyfikacja JMESPath

S3: kubełki, obiekty, uprawnienia

Poziom: podstawy **Dotyczy: Amazon S3, AWS CLI v2** **Czas: ~30 min**

S3 wygląda jak dysk sieciowy z folderami, ale nim nie jest — ta strona tłumaczy, czym naprawdę jest kubełek i obiekt, zanim przejdziemy do komend, którymi się nimi zarządza.

Komendy wykonujemy z CLI bash ze skonfigurowanym profilem AWS CLI, opisanym na stronie AWS CLI: instalacja, profile, poświadczenia.

Skróty użyte na tej stronie (10)

Skrót	Co znaczy
API	Application Programming Interface: sposób, w jaki programy rozmawiają ze sobą
AWS	Amazon Web Services: chmura Amazonu
CLI	Command Line Interface: praca przez wpisywanie komend w terminalu
IAM	Identity and Access Management: zarządzanie tożsamościami i uprawnieniami (w AWS: użytkownicy, role, polityki)
JSON	JavaScript Object Notation: tekstowy format danych z klamrami i cudzysłowami
KMS	Key Management Service: usługa chmurowa przechowująca i zarządzająca kluczami szyfrującymi
PDF	Portable Document Format: format dokumentu, który wygląda tak samo na każdym komputerze, np. zakres akredytacji jednostki na stronie PCA
REST	sposób budowania API przez zwykłe żądania HTTP (GET, POST) zamiast osobnego protokołu
S3	Simple Storage Service: magazyn plików w chmurze AWS
URL	Uniform Resource Locator: adres strony lub zasobu

Fiszka

TYLKO ODCZYT

Wylistowanie wszystkich kubełków na koncie: aws s3 ls

```
aws s3 ls
```

Bez podanej ścieżki `aws s3 ls` zwraca listę wszystkich kubełków widocznych na koncie, z datą utworzenia każdego z nich.

Przykład:

```
2024-03-11 09:22:15 przyklad-backup
2025-01-08 14:03:47 przyklad-logi-aplikacji
2025-06-02 11:47:03 przyklad-statyczne-pliki
```

aws = AWS CLI: narzędzie chmury Amazonu

TYLKO ODCZYT

Wylistowanie zawartości kubełka: aws s3 ls

```
aws s3 ls s3://NazwaKubelka/
```

Zwraca obiekty i „foldery" (w rzeczywistości wspólne prefiksy) leżące bezpośrednio pod wskazanym kluczem — jeden poziom w dół, nie rekurencyjnie.

Przykład:

```
                           PRE faktury/
2026-01-15 08:12:33     481293 raport-styczen.csv
```

aws = AWS CLI: narzędzie chmury Amazonu

TYLKO ODCZYT

Rekurencyjne wylistowanie całej zawartości: aws s3 ls --recursive

```
aws s3 ls s3://NazwaKubelka/ --recursive
```

Dodanie `--recursive` schodzi we wszystkie prefiksy naraz i zwraca pełną, płaską listę kluczy — przydatne, gdy chcemy zobaczyć rzeczywistą liczbę i rozmiar obiektów, a nie tylko

strukturę pierwszego poziomu.

Przykład:

```
2026-01-10 07:00:01     102934 faktury/2026/styczen/f-001.pdf
2026-01-11 07:00:02      98211 faktury/2026/styczen/f-002.pdf
2026-01-15 08:12:33     481293 raport-styczen.csv
```

aws = AWS CLI: narzędzie chmury Amazonu

ZMIENIA SYSTEM

Skopiowanie pojedynczego pliku do kubełka: aws s3 cp

```
aws s3 cp SciezkaDoPliku s3://NazwaKubelka/raport-styczen.csv
```

Kopiujemy jeden plik lokalny pod wskazany klucz w kubełku — działa symetrycznie w drugą stronę (`aws s3 cp s3://NazwaKubelka/raport-styczen.csv SciezkaDoPliku` pobiera obiekt lokalnie).

Przykład:

```
upload: ./raport-styczen.csv to s3://przyklad-backup/raport-styczen.csv
```

aws = AWS CLI: narzędzie chmury Amazonu

TYLKO ODCZYT

Synchronizacja katalogu lokalnego z kubełkiem: aws s3 sync

```
aws s3 sync SciezkaDoPliku s3://NazwaKubelka/faktury/
```

> **Pojęcie: `sync`.** Kopiuje tylko te pliki, których nie ma jeszcze w celu albo które zmieniły rozmiar lub datę modyfikacji — w odróżnieniu od `cp`, które kopiuje zawsze, `sync` porównuje źródło i cel przed każdym transferem. Domyślnie `sync` nigdy nie usuwa plików po stronie docelowej, nawet jeśli zniknęły ze źródła.

Przykład:

```
upload: faktury/f-003.pdf to s3://przyklad-backup/faktury/f-003.pdf
upload: faktury/f-004.pdf to s3://przyklad-backup/faktury/f-004.pdf
```

aws = AWS CLI: narzędzie chmury Amazonu

OSTROŻNIE · NIEODWRACALNE

Synchronizacja z usunięciem plików osieroconych: sync --delete

```
aws s3 sync SciezkaDoPliku s3://NazwaKubelka/faktury/ --delete
```

Flaga `--delete` dodatkowo usuwa z kubełka te obiekty, których nie ma już w źródle — dokładne lustrzane odbicie katalogu lokalnego, przydatne przy publikowaniu statycznej strony, gdzie nie chcemy zostawiać nieaktualnych plików.

Uważaj: `--delete` jest nieodwracalne bez włączonego wersjonowania kubełka — usunięty obiekt bez wersjonowania znika bezpowrotnie.

aws = AWS CLI: narzędzie chmury Amazonu

TYLKO ODCZYT

Synchronizacja z pominięciem wybranych plików: sync --exclude

```
aws s3 sync SciezkaDoPliku s3://NazwaKubelka/faktury/ --exclude "*.tmp"
```

Wzorzec podany w `--exclude` pomija pasujące pliki podczas synchronizacji — działa na ścieżce względem katalogu źródłowego, nie na samej nazwie pliku.

aws = AWS CLI: narzędzie chmury Amazonu

TYLKO ODCZYT

Synchronizacja z wyjątkiem od wykluczenia: sync --exclude i --include razem

```
aws s3 sync SciezkaDoPliku s3://NazwaKubelka/faktury/ --exclude "*" --include "*.pdf"
```

> **Pojęcie: kolejność `--exclude` / `--include`.** AWS CLI przetwarza filtry w kolejności podania w komendzie, każdy kolejny nadpisując decyzję poprzedniego dla pasujących plików. `--exclude "*"` odrzuca wszystko, a następujący po nim `--include "*.pdf"` przywraca do synchronizacji tylko pliki PDF — odwrócenie kolejności tych dwóch flag da inny, zwykle niezamierzony wynik.

AWS = Amazon Web Services: chmura Amazonu · **CLI** = Command Line Interface: praca przez wpisywanie komend w terminalu · **PDF** = Portable Document Format: format dokumentu, który wygląda tak samo na każdym komputerze, np. zakres akredytacji jednostki na stronie PCA · **aws** = AWS CLI: narzędzie chmury Amazonu

ZMIENIA SYSTEM

Przeniesienie obiektu w obrębie S3: aws s3 mv

```
aws s3 mv s3://NazwaKubelka/raport-styczen.csv s3://NazwaKubelka/archiwum/raport-styczen.csv
```

Wewnętrznie wykonuje kopię pod nowym kluczem i usunięcie starego — nie jest to operacja atomowa jednego zasobu, tylko dwie operacje wykonane jedna po drugiej.

Przykład:

```
move: s3://przyklad-backup/raport-styczen.csv to s3://przyklad-backup/archiwum/raport-styczen.csv
```

S3 = Simple Storage Service: magazyn plików w chmurze AWS · **aws** = AWS CLI: narzędzie chmury Amazonu

OSTROŻNIE · NIEODWRACALNE

Usunięcie pojedynczego obiektu: aws s3 rm

```
aws s3 rm s3://NazwaKubelka/archiwum/raport-styczen.csv
```

Usuwa wskazany obiekt. Bez włączonego wersjonowania usunięcie jest natychmiastowe i ostateczne.

Przykład:

```
delete: s3://przyklad-backup/archiwum/raport-styczen.csv
```

□□ **aws** = AWS CLI: narzędzie chmury Amazonu

□□ OSTROŻNIE · NIEODWRACALNE

Usunięcie całego prefiksu: aws s3 rm --recursive

```
aws s3 rm s3://NazwaKubelka/archiwum/ --recursive
```

Bez `--recursive` `aws s3 rm` na kluczu kończącym się ukośnikiem nie usuwa niczego poza ewentualnym pustym obiektem-znacznikiem — usunięcie całej zawartości „folderu" wymaga jawnego rekurencyjnego przejścia po wszystkich pasujących kluczach.

□□ **aws** = AWS CLI: narzędzie chmury Amazonu

□□ TYLKO ODCZYT

Wygenerowanie tymczasowego linku do obiektu: aws s3 presign

```
aws s3 presign s3://NazwaKubelka/faktury/2026/styczen/f-001.pdf --expires-in 3600
```

> **Pojęcie: presigned URL.** Adres URL zawierający w parametrach zapytania podpis kryptograficzny wygenerowany naszymi poświadczeniami — pozwala dowolnej osobie z tym linkiem pobrać (albo, przy odpowiednim wywołaniu, wgrać) konkretny obiekt bez posiadania własnych poświadczeń AWS, przez ograniczony czas określony flagą `--expires-in` (w sekundach). Przydatne, gdy chcemy udostępnić jeden plik z prywatnego kubełka bez zmiany jego uprawnień na trwałe.

Przykład:

```
https://przyklad-backup.s3.eu-central-1.amazonaws.com/faktury/2026/styczen/f-001.pdf?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIOSFODNN7EXAMPLE%2F20260125%2Feu-central-1%2Fs3%2Faws4_request&X-Amz-Date=20260125T090000Z&X-Amz-Expires=3600&X-Amz-Signature=8f7b2a...
```

Uważaj: presigned URL działa dla dowolnej osoby, która go otrzyma, aż do wygaśnięcia — traktujemy go jak hasło jednorazowe, nie wklejamy do kanałów, które ktoś niepowołany może przeczytać.

□□ **URL** = Uniform Resource Locator: adres strony lub zasobu · **AWS** = Amazon Web Services: chmura Amazonu · **aws** = AWS CLI: narzędzie chmury Amazonu

Z życia wzięte

Z ŻYCIA WZIĘTE

O 14:20 ktoś w zespole próbuje „zmienić nazwę folderu" faktury w konsoli AWS i po dziesięciu minutach pyta Tomka, dlaczego operacja na milionie plików wciąż się kręci, a licznik kosztów rośnie. Tomek tłumaczy, że w S3 nie ma prawdziwych katalogów, więc zmiana nazwy prefiksu to w rzeczywistości skopiowanie i usunięcie każdego z tych plików osobno, nie jedna operacja na strukturze. Zespół przerywa zadanie, dzieli je na mniejsze partie uruchamiane w tle i od tej pory planuje duże przenosiny prefiksów poza godziny szczytu.

Czym jest kubełek i obiekt

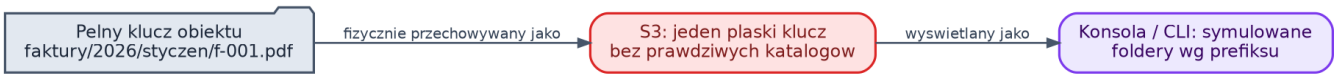
Pojęcie: kubełek (bucket). Kontener najwyższego poziomu w S3 — jednostka, do której przypisujemy region, uprawnienia, ustawienia wersjonowania i szyfrowania. Nazwa kubełka jest globalnie unikatowa w obrębie całego AWS, nie tylko naszego konta — raz zajętej nazwy nie da się użyć ponownie, dopóki poprzedni właściciel jej nie zwolni.

Pojęcie: obiekt (object). Pojedynczy plik przechowywany w kubełku, wraz z metadanymi (typ zawartości, znaczniki, nagłówki cache). Obiekt jest identyfikowany przez klucz — pełną „ścieżkę" w obrębie kubełka, np. `faktury/2026/styczen/f-001.pdf`.

Pojęcie: klucz (key). Ciąg znaków jednoznacznie identyfikujący obiekt wewnątrz kubełka. Klucz może zawierać ukośniki i konsola AWS wyświetla je jak strukturę katalogów, ale to złudzenie interfejsu — S3 nie ma pojęcia katalogu. `faktury/2026/styczen/f-001.pdf` to dla S3 jeden płaski klucz, nie trzy zagnieżdżone foldery.

Dlaczego S3 nie jest systemem plików

Mimo ukośników w kluczach i mimo że konsola i CLI potrafią symulować przechodzenie po „folderach", S3 to magazyn typu obiekt-klucz-wartość (object storage), nie system plików:



Klucz obiektu jest fizycznie jednym płaskim ciągiem znaków, a foldery widziane w konsoli czy CLI to tylko symulacja oparta na prefiksie (kliknięcie otwiera pełny rozmiar)

- Nie ma operacji „zmień nazwę katalogu" ani „przenieś podkatalog" jako operacji atomowej — zmiana klucza wymaga skopiowania obiektu pod nowym kluczem i usunięcia starego, co przy dużej liczbie obiektów pod wspólnym prefiksem oznacza tyle operacji, ile obiektów.
- Nie ma blokad plikowych ani jednoczesnego zapisu przyrostowego — obiekt zapisuje się w całości albo wcale, nie da się dopisać bajtów na końcu istniejącego obiektu tak jak do otwartego pliku.
- Pusty „folder" w konsoli AWS to w rzeczywistości osobny, zerobajtowy obiekt z kluczem kończącym się ukośnikiem — usunięcie go nie usuwa niczego poza tym jednym obiektem, w szczególności nie usuwa obiektów, których klucze zaczynają się tym samym prefiksem.

Konsekwencja praktyczna: kiedy myślimy o „katalogu" w S3, myślimy naprawdę o prefiksie klucza — wspólnym początku wielu kluczy, po którym filtrujemy i grupujemy, a nie o realnym obiekcie hierarchii.

Klasy magazynowania

> **Pojęcie: klasa magazynowania (storage class).** Wariant przechowywania tego samego obiektu, różniący się ceną za gigabajt, czasem dostępu do danych i minimalnym okresem przechowywania rozliczanym niezależnie od tego, czy obiekt faktycznie tak długo leżał. Wybór klasy jest kompromisem między kosztem przechowywania a kosztem i czasem odczytu.

Klasa	Czas dostępu	Typowe zastosowanie
`STANDARD`	natychmiastowy	dane używane regularnie
`STANDARD_IA` (Infrequent Access)	natychmiastowy, wyższa opłata za odczyt	dane rzadko czytane, ale wymagające natychmiastowego dostępu przy potrzebie
`INTELLIGENT_TIERING`	natychmiastowy	dane o nieprzewidywalnym wzorcu dostępu — S3 samo przenosi obiekt między warstwami
`GLACIER`	od minut do godzin (w zależności od trybu odzyskania)	archiwum, rzadkie odtworzenie

Klasa	Czas dostępu	Typowe zastosowanie
`GLACIER_DEEP_ARCHIVE`	do 12 godzin	archiwum długoterminowe, np. wymogi regulacyjne

Reguły cyklu życia

> **Pojęcie: reguła cyklu życia (lifecycle rule).** Zasada zapisana na poziomie kubełka, automatycznie przenosząca obiekty między klasami magazynowania albo usuwająca je po upływie zadanego czasu od utworzenia, bez ręcznej interwencji. Typowa reguła: obiekty w prefiksie `logi/` starsze niż 30 dni przenosimy do `STANDARD_IA`, starsze niż 90 dni do `GLACIER`, starsze niż 365 dni usuwamy. Reguły cyklu życia są podstawowym sposobem trzymania kosztów magazynu pod kontrolą bez ręcznego przeglądania zawartości kubełka.

Kiedy `aws s3` nie wystarcza — `aws s3api`

> **Pojęcie: `aws s3` kontra `aws s3api`.** `aws s3` to zestaw poleceń wysokopoziomowych, zaprojektowanych pod kątem wygody (kopiowanie, synchronizacja) i ukrywających część szczegółów API. `aws s3api` to bezpośrednie odwzorowanie każdego wywołania REST API S3 jeden do jednego — mniej wygodne w codziennej pracy, ale jedyna droga do operacji, których `aws s3` nie udostępnia: zarządzania politykami, wersjonowaniem, ACL-ami i metadanymi na poziomie pojedynczych wersji obiektów.

Odczytanie polityki kubełka: aws s3api get-bucket-policy

```
aws s3api get-bucket-policy --bucket NazwaKubelka --query Policy --output text
```

Zwraca aktualnie obowiązującą politykę dostępu przypisaną do kubełka w postaci dokumentu JSON — `--query Policy --output text` wyciąga sam dokument polityki bez otaczającej go koperty odpowiedzi API.

Przykład:

```
{"Version":"2012-10-17","Statement":[{"Sid":"AllowReadOnly","Effect":"Allow","Principal":{"AWS":"arn:aws:iam::123456789012:user/nazwa_usera"},"Action":["s3:GetObject","s3:ListBucket"],"Resource":["arn:aws:s3:::przyklad-backup","arn:aws:s3:::przyklad-backup/*"]}]}
```

Włączenie wersjonowania kubełka: aws s3api put-bucket-versioning

```
aws s3api put-bucket-versioning --bucket NazwaKubelka --versioning-configuration Status=Enabled
```

> **Pojęcie: wersjonowanie (versioning).** Mechanizm, dzięki któremu nadpisanie albo usunięcie obiektu nie kasuje poprzedniej zawartości, tylko tworzy nową wersję pod tym samym kluczem, zachowując poprzednie wersje dostępne do odczytu i przywrócenia. Usunięcie obiektu w kubełku z włączonym wersjonowaniem nie usuwa danych — dodaje znacznik usunięcia (delete marker) jako najnowszą wersję, a starsze wersje wciąż leżą w kubełku, dopóki nie usunie się ich jawnie po identyfikatorze wersji.

Włączamy wersjonowanie przede wszystkim z dwóch powodów: ochrony przed przypadkowym nadpisaniem albo usunięciem (błąd operatora, błąd w skrypcie z `--delete`) oraz jako wymogu zgodności w regulacjach wymagających zachowania historii zmian dokumentu. Kosztem jest rosnące zużycie magazynu — każda nadpisana wersja nadal zajmuje miejsce, dopóki nie usunie jej reguła cyklu życia dedykowana starym wersjom.

Przykład:

```
{
    "ResponseMetadata": {
        "HTTPStatusCode": 200
    }
}
```

Wylistowanie wszystkich wersji obiektów: aws s3api list-object-versions

```
aws s3api list-object-versions --bucket NazwaKubelka --prefix faktury/2026/styczen/
```

Zwraca pełną historię wersji i znaczników usunięcia dla obiektów pod wskazanym prefiksem — jedyny sposób, żeby zobaczyć wersje inne niż najnowsza, bo `aws s3 ls` pokazuje zawsze tylko bieżący stan.

Przykład:

```
{
    "Versions": [
        {
            "Key": "faktury/2026/styczen/f-001.pdf",
            "VersionId": "3sL4kqtJlcpXroDTDmJ+rmSpXd3dIbrHY+MTRCxf3vjVBH40Nr8X8gdRQBpUMLUo",
            "IsLatest": true,
            "LastModified": "2026-01-15T08:12:33.000Z",
            "Size": 102934
        },
        {
            "Key": "faktury/2026/styczen/f-001.pdf",
            "VersionId": "QUpfdndhfd8438MNFDN93jdnFkRFi9gsdzZFrN8gTsvBQ1KyMKp",
            "IsLatest": false,
            "LastModified": "2026-01-10T07:00:01.000Z",
            "Size": 101822
        }
    ]
}
```

Szyfrowanie po stronie serwera

> **Pojęcie: szyfrowanie po stronie serwera (Server-Side Encryption, SSE).** Szyfrowanie danych w spoczynku wykonywane przez S3 automatycznie przy zapisie i odszyfrowywane automatycznie przy odczycie autoryzowanym żądaniem — klient nie musi implementować własnej kryptografii, żeby dane na dysku dostawcy nie leżały jawnym tekstem. S3 udostępnia trzy warianty: `SSE-S3` (klucze zarządzane w całości przez AWS, najprostszy, domyślny dla nowych kubełków), `SSE-KMS` (klucze zarządzane przez usługę KMS, z możliwością audytu użycia klucza i rotacji), oraz `SSE-C` (klucz dostarcza klient przy każdym żądaniu, AWS go nie przechowuje). Wybór wariantu jest kompromisem między prostotą (`SSE-S3`) a kontrolą i audytowalnością (`SSE-KMS`).

Szyfrowanie po stronie serwera chroni dane leżące na dysku dostawcy przed dostępem do surowego nośnika — nie zastępuje kontroli dostępu opisanej politykami IAM i politykami kubełka na

stronie Konto i uprawnienia do pobierania plików z S3, bo autoryzowane żądanie zawsze otrzymuje dane odszyfrowane.

Częste błędy

Traktowanie prefiksu jak prawdziwego katalogu przy dużej liczbie obiektów. Zmiana nazwy „folderu" zawierającego milion kluczy oznacza milion operacji kopiowania i usuwania, nie jedną operację zmiany nazwy — czas i koszt tej operacji rośnie liniowo z liczbą obiektów, nie jest stały.

Użycie `sync --delete` bez wcześniejszego `--dryrun`. `--dryrun` pokazuje, co komenda by zrobiła, bez faktycznego wykonania transferu — pominięcie tego kroku przy pierwszym uruchomieniu `sync --delete` na nowo skonfigurowanym źródle bywa przyczyną nieodwracalnej utraty danych w kubełku bez wersjonowania.

Zakładanie, że usunięcie obiektu w kubełku z wersjonowaniem faktycznie zwalnia miejsce. Dopóki nie usunie się jawnie starych wersji (ręcznie albo regułą cyklu życia), obiekt „usunięty" nadal zajmuje magazyn pod poprzednimi identyfikatorami wersji.

□□ Ćwiczenia

> Zadania wykonujemy na testowym kubełku, najlepiej pustym i przeznaczonym wyłącznie do ćwiczeń.

1. Wyjaśniamy, dlaczego S3 nie jest systemem plików, mimo że konsola AWS pokazuje jego zawartość jako drzewo folderów.
2. Tworzymy lokalny katalog z kilkoma plikami, synchronizujemy go do testowego kubełka komendą `sync`, usuwamy jeden plik lokalnie i sprawdzamy różnicę między ponownym `sync` bez i z flagą `--delete`.
3. Włączamy wersjonowanie na testowym kubełku, nadpisujemy ten sam obiekt dwa razy i komendą `list-object-versions` sprawdzamy, ile wersji faktycznie tam leży.
4. Generujemy presigned URL ważny 60 sekund, otwieramy go w przeglądarce od razu i ponownie po minucie — opisujemy różnicę w zachowaniu.

Powiązane: AWS CLI: instalacja, profile, poświadczenia · Konto i uprawnienia do pobierania plików z S3 · Pobieranie plików z S3 przez PowerShell · Backup bazy do S3: agent backint i jego ograniczenia

Źródła: dokumentacja AWS CLI, sekcja `s3` i `s3api` · dokumentacja Amazon S3, rozdziały o klasach magazynowania, wersjonowaniu i szyfrowaniu po stronie serwera

Konto i uprawnienia do pobierania plików z S3

Typ: procedura **Dotyczy: AWS IAM, Amazon S3** **Czas: ~20 min**

Efekt: istnieje użytkownik IAM z kluczami dostępowymi, który potrafi wyłącznie odczytywać obiekty z jednego, wskazanego kubełka i prefiksu — nic więcej, żadnego zapisu, żadnego dostępu do pozostałych kubełków na koncie.

Skróty użyte na tej stronie (4)

Skrót	Co znaczy
AWS	Amazon Web Services: chmura Amazonu
EC2	Elastic Compute Cloud: usługa AWS wynajmująca gotowe maszyny wirtualne na godziny
IAM	Identity and Access Management: zarządzanie tożsamościami i uprawnieniami (w AWS: użytkownicy, role, polityki)
S3	Simple Storage Service: magazyn plików w chmurze AWS

Zanim zaczniemy

Z ŻYCIA WZIĘTE

O 15:30 kontrahent zewnętrzny zgłasza, że komendą `aws s3 ls` widzi zawartość zupełnie innego kubełka niż ten, do którego miał dostać dostęp tydzień wcześniej. Tomek sprawdza politykę użytkownika i widzi, że jest poprawnie zawężona do jednego prefiksu, ale ktoś wcześniej dopisał szeroką politykę bezpośrednio na samym kubełku, żeby „na chwilę ułatwić sobie testy". Usuwa politykę kubełka i zostawia wyłącznie zawężoną politykę użytkownika, po czym kontrahent znowu widzi tylko to, co powinien.

Wymaganie	Uwaga
Dostęp do konsoli albo CLI z uprawnieniami do IAM	tworzenie użytkowników, polityk i kluczy wymaga uprawnień administracyjnych w IAM, nie samego dostępu do S3

Wymaganie	Uwaga
Ustalona nazwa kubełka i prefiksu	konto ma dostać dostęp do jednego kubełka i — jeśli to możliwe — jednego prefiksu wewnątrz niego, nie do całego konta
Zdecydowany sposób dostarczenia kluczy	klucze trzeba przekazać stronie odbierającej pliki jednym bezpiecznym kanałem, nigdy przez e-mail w postaci jawnej

> **Pojęcie: użytkownik IAM kontra rola.** Użytkownik IAM ma poświadczenia (hasło do konsoli albo klucze dostępowe), które istnieją niezależnie od tego, co je aktualnie używa — pasują do skryptu, do zewnętrznej firmy, do dowolnego procesu spoza AWS. Rola nie ma własnych stałych poświadczeń — jest przyjmowana tymczasowo przez coś, co już ma tożsamość w AWS (inny użytkownik, instancja EC2, konto z innej organizacji), opisana szerzej na stronie AWS CLI: instalacja, profile, poświadczenia. Do pobierania plików przez zewnętrzny, niekontrolowany przez nas proces (kontrahenta, zewnętrzny skrypt, integrację poza AWS) wybieramy użytkownika z kluczami — rola nie ma jak zostać „przyjęta" przez coś, co nie jest częścią naszego środowiska AWS.

Uważaj: klucz dostępu (Access Key) i klucz tajny (Secret Access Key) są widoczne do skopiowania tylko raz, w momencie utworzenia — po zamknięciu tego widoku sekretu nie da się już odzyskać, można jedynie unieważnić klucz i wygenerować nowy.

Przebieg

1. Utworzenie użytkownika IAM

```
aws iam create-user --user-name NazwaUsera
```

Tworzymy użytkownika bez żadnych uprawnień — same uprawnienia nadamy w kolejnym kroku, dołączając do niego politykę.

Przykład:

```
{
    "User": {
        "UserName": "NazwaUsera",
        "UserId": "AIDACKCEVSQ6C2EXAMPLE",
        "Arn": "arn:aws:iam::123456789012:user/NazwaUsera",
        "CreateDate": "2026-01-20T09:12:00Z"
```

```
    }
}
```

2. Utworzenie polityki minimalnych uprawnień

> **Pojęcie: zasada minimalnych uprawnień (least privilege).** Nadawanie dokładnie tego zakresu dostępu, który jest potrzebny do wykonania zadania — ani jednego uprawnienia więcej. Dla konta służącego wyłącznie do pobierania plików oznacza to prawo odczytu (`s3:GetObject`) i prawo wylistowania zawartości (`s3:ListBucket`), ograniczone do jednego kubełka i jednego prefiksu wewnątrz niego — nigdy `s3:*` na całym koncie.

Zapisujemy poniższy dokument do pliku lokalnego:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ListujTylkoWskazanyPrefiks",
      "Effect": "Allow",
      "Action": "s3:ListBucket",
      "Resource": "arn:aws:s3:::NazwaKubelka",
      "Condition": {
        "StringLike": {
          "s3:prefix": "NazwaPrefiksu/*"
        }
      }
    },
    {
      "Sid": "CzytajTylkoWskazanyPrefiks",
      "Effect": "Allow",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::NazwaKubelka/NazwaPrefiksu/*"
    }
  ]
}
```

> (gdzie `NazwaKubelka` to nazwa kubełka, a `NazwaPrefiksu` to katalog logiczny wewnątrz niego, do którego ograniczamy dostęp — warunek `s3:prefix` w pierwszym bloku jest konieczny, bo bez niego `s3:ListBucket` bez warunku pozwoliłoby wylistować całą zawartość kubełka, nawet jeśli `s3:GetObject` i tak nie pozwoliłoby niczego z niej pobrać poza wskazanym prefiksem)

Tworzymy z tego pliku politykę zarządzaną:

```
aws iam create-policy --policy-name PolitykaOdczytuNazwaPrefiksu --policy-document
file://SciezkaDoPliku
```

Przykład:

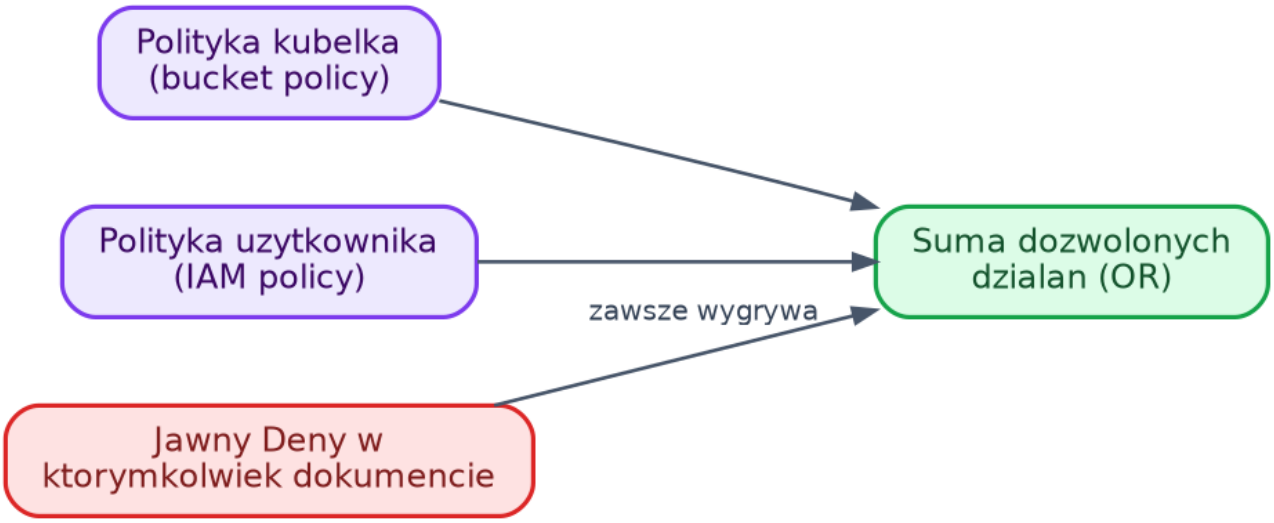
```
{
    "Policy": {
        "PolicyName": "PolitykaOdczytuNazwaPrefiksu",
        "Arn": "arn:aws:iam::123456789012:policy/PolitykaOdczytuNazwaPrefiksu",
        "CreateDate": "2026-01-20T09:15:00Z"
    }
}
```

3. Podpięcie polityki do użytkownika

```
aws iam attach-user-policy --user-name NazwaUsera --policy-arn
arn:aws:iam::123456789012:policy/PolitykaOdczytuNazwaPrefiksu
```

Polecenie przy powodzeniu nic nie zwraca — brak odpowiedzi jest tu oznaką sukcesu, nie błędu.

> **Pojęcie: polityka kubełka kontra polityka użytkownika.** Polityka kubełka (bucket policy) jest przypisana do zasobu i mówi, kto może z nim coś robić. Polityka użytkownika (jak ta utworzona w kroku 2) jest przypisana do tożsamości i mówi, co ta tożsamość może robić. AWS łączy oba dokumenty przy każdym żądaniu i domyślnie wynikiem jest suma dozwolonych działań (logiczne OR) — z jednym wyjątkiem, który zmienia wszystko: jawne `"Effect": "Deny"` w którymkolwiek z dwóch dokumentów zawsze wygrywa, niezależnie od tego, co pozwala drugi dokument. Jeśli polityka kubełka blokuje dostęp z adresów spoza firmowej sieci, żadna polityka użytkownika tego nie obejdzie.



AWS sumuje dozwolone działania z polityki kubełka i polityki użytkownika, ale jawny Deny w którymkolwiek dokumencie zawsze wygrywa (kliknięcie otwiera pełny rozmiar)

4. Utworzenie kluczy dostępu

```
aws iam create-access-key --user-name NazwaUsera
```

Generuje parę klucz dostępu / klucz tajny — jedyny moment, w którym klucz tajny jest widoczny w odpowiedzi.

Przykład:

```
{
    "AccessKey": {
        "UserName": "NazwaUsera",
        "AccessKeyId": "AKIAQZXK4EXAMPLEKEY",
        "SecretAccessKey": "7MDENG/bPxRfiCYEXAMPLEKEYwJalrXUtnFEMI/K",
        "Status": "Active",
        "CreateDate": "2026-01-20T09:18:00Z"
    }
}
```

Zapisujemy oba klucze od razu w bezpiecznym miejscu (menedżer haseł zespołu) — ten widok się nie powtórzy.

5. Skonfigurowanie profilu odbierającego pliki

Stronie odbierającej pliki przekazujemy oba klucze bezpiecznym kanałem, a nie samą komendę — sposób skonfigurowania profilu opisuje strona AWS CLI: instalacja, profile, poświadczenia:

```
aws configure --profile NazwaProfilu
```

Weryfikacja

Sprawdzamy z perspektywy nowo utworzonego użytkownika, czy widzi dokładnie to, co powinien — ani więcej, ani mniej:

```
aws s3 ls s3://NazwaKubelka/NazwaPrefiksu/ --profile NazwaProfilu
```

Przykład powodzenia:

```
2026-01-15 08:12:33     102934 f-001.pdf
2026-01-16 08:05:11      98211 f-002.pdf
```

Sprawdzamy też, że dostęp poza wskazany prefiks jest odrzucony:

```
aws s3 ls s3://NazwaKubelka/InnyPrefiks/ --profile NazwaProfilu
```

Przykład prawidłowego odrzucenia:

```
An error occurred (AccessDenied) when calling the ListObjectsV2 operation: Access Denied
```

Jeśli to drugie wywołanie *nie* zwraca błędu, polityka jest zbyt szeroka — wracamy do kroku 2 i zawężamy warunek `s3:prefix`.

Wycofanie zmian

Rotacja albo cofnięcie dostępu nie wymaga usuwania całego użytkownika — najpierw dezaktywujemy klucz:

```
aws iam update-access-key --user-name NazwaUsera --access-key-id AKIAQZXK4EXAMPLEKEY --status Inactive
```

Dezaktywowany klucz przestaje działać natychmiast, ale zostaje w koncie do ewentualnego przywrócenia — dopiero po potwierdzeniu, że nic go już nie używa, usuwamy go trwale:

```
aws iam delete-access-key --user-name NazwaUsera --access-key-id AKIAQZXK4EXAMPLEKEY
```

> **Pojęcie: rotacja kluczy.** Regularna wymiana klucza dostępowego na nowy, nawet jeśli nic nie wskazuje na jego wyciek — ograniczenie okna czasowego, w którym ewentualnie skompromitowany klucz jest użyteczny. Rotację wykonujemy generując nowy klucz *przed* dezaktywacją starego (użytkownik IAM może mieć jednocześnie dwa aktywne klucze), podmieniając go po stronie odbiorcy, a dopiero potem dezaktywując i usuwając poprzedni.

Całkowite cofnięcie dostępu, gdy konto nie jest już potrzebne:

```
aws iam detach-user-policy --user-name NazwaUsera --policy-arn arn:aws:iam::123456789012:policy/PolitykaOdczytuNazwaPrefiksu
```

```
aws iam delete-user --user-name NazwaUsera
```

`delete-user` kończy się błędem, dopóki użytkownik ma odpięte nie wszystkie polityki i niekasnięte klucze dostępu — AWS celowo nie pozwala usunąć tożsamości z aktywnymi poświadczeniami czy uprawnieniami w jednym kroku.

Gdy coś nie zadziała

Objaw	Przyczyna	Naprawa
`AccessDenied` przy `ListBucket` mimo poprawnej polityki	brak warunku `s3:prefix` albo literówka w nazwie prefiksu	porównujemy wartość w `Condition` z rzeczywistą nazwą prefiksu w kubełku
`AccessDenied` przy `GetObject`, choć `ListBucket` działa	zasób w polityce wskazuje kubełek bez `/*` na końcu	poprawiamy `Resource` na `arn:aws:s3:::NazwaKubelka/NazwaPrefiksu/*`
Dostęp działa mimo restrykcyjnej polityki użytkownika	polityka kubełka (bucket policy) pozwala szerzej i nic jej nie ogranicza	sprawdzamy `aws s3api get-bucket-policy --bucket NazwaKubelka` i dodajemy tam jawny warunek albo `Deny`
`An error occurred (InvalidClientTokenId)`	klucz dostępu został wcześniej dezaktywowany albo usunięty	generujemy nowy klucz krokiem 4 i aktualizujemy profil po stronie odbiorcy

Objaw	Przyczyna	Naprawa
`aws configure` zapisuje dane, ale komendy i tak używają innego konta	zmienna `AWS_PROFILE` w tle wskazuje inny profil niż ten, który właśnie konfigurujemy	sprawdzamy `echo $AWS_PROFILE` i jawnie podajemy `--profile NazwaProfilu` przy weryfikacji

Powiązane: AWS CLI: instalacja, profile, poświadczenia · S3: kubełki, obiekty, uprawnienia · Pobieranie plików z S3 przez PowerShell

Źródła: dokumentacja AWS IAM, sekcja „Access keys" i „Policies" · dokumentacja Amazon S3, sekcja „Bucket policy and user policy"

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`

Backup bazy do S3: agent backint i jego ograniczenia

Poziom: podstawy **Dotyczy: SAP HANA, agent Backint dla AWS S3** **Czas: ~15 min**

Skróty użyte na tej stronie (5)

Skrót	Co znaczy
AWS	Amazon Web Services: chmura Amazonu
HANA	SAP HANA: baza danych w pamięci firmy SAP, używana do dużych systemów księgowych i magazynowych
HTTP	Hypertext Transfer Protocol: protokół stron WWW
S3	Simple Storage Service: magazyn plików w chmurze AWS
SAP	Systeme, Anwendungen und Produkte in der Datenverarbeitung: niemiecki producent oprogramowania klasy ERP, właściciel silnika HANA i przejętego wcześniej Sybase ASE

Z życia wzięte

Z ŻYCIA WZIĘTE

O 4:10 nocne zadanie backupu bazy HANA kończy się błędem `Error reading backup from BACKINT`, a rano trzeba wyjaśnić, czy dane sprzed nocy są w ogóle bezpieczne. Tomek sprawdza w S3, czy oczekiwany klucz backupu w ogóle istnieje, i widzi, że plik jest, ale ma połowę spodziewanego rozmiaru, bo transfer do S3 urwał się w połowie przez chwilową awarię sieci. Uruchamia backup ad hoc jeszcze raz w ciągu dnia i dopiero po potwierdzeniu pełnego rozmiaru pliku w S3 uznaje kopię za odtwarzalną.

Czym jest backint

> **Pojęcie: backint.** Interfejs programowy SAP HANA (Backup Interface for SAP HANA), przez który silnik bazy danych oddaje strumień kopii zapasowej zewnętrznemu narzędziu zamiast zapisywać go bezpośrednio w systemie plików. HANA nie wie i nie musi wiedzieć, gdzie backint faktycznie zapisuje dane — może to być deduplikator, biblioteka backupowa albo, jak w tym przypadku, agent kierujący strumień wprost do magazynu obiektowego (S3). Z perspektywy HANA backup przez backint wygląda identycznie jak backup do pliku — różni się tylko tym, co dzieje się po drugiej stronie interfejsu.



HANA oddaje strumień backupu przez interfejs backint agentowi, który zapisuje albo odczytuje dane z magazynu S3 (kliknięcie otwiera pełny rozmiar)

Backup pełny, różnicowy i przyrostowy działają przez backint tak samo jak przez system plików — różnicę między tymi rodzajami backupu, niezależną od tego, gdzie trafia strumień, opisuje strona Backup pełny, różnicowy, przyrostowy, logi. Backint dokłada do tego wyłącznie warstwę transportu do zewnętrznego magazynu.

Zanim backup w ogóle trafi do S3, HANA musi wiedzieć, że ma go oddać przez backint, a nie zapisać na lokalny system plików — wybór ten ustawiamy w parametrze `catalog_backup_using_backint` (dla katalogu backupu) i analogicznych parametrach dla backupu danych i logów w `global.ini`, w sekcji `[backup]`. Sam agent backint dla S3 jest osobnym programem zewnętrznym, wskazanym w konfiguracji HANA ścieżką do binarki i do jego własnego pliku parametrów — to właśnie w tym pliku parametrów agenta leżą `MaximumConcurrentFilesForRestore` i `DownloadConcurrency` opisane niżej, nie w konfiguracji samej HANA.

Limity agenta backint dla S3

Agent backint kierujący dane do S3 ma własne, konfigurowalne limity równoległości, niezależne od limitów samego S3 czy przepustowości łącza — te limity decydują o tym, ile plików backupu agent przesyła (albo odzyskuje) naraz.

> **Pojęcie: `MaximumConcurrentFilesForRestore`.** Parametr konfiguracji agenta określający, ile plików backupu może być jednocześnie odczytywanych z S3 podczas operacji odtwarzania. Wyższa wartość przyspiesza odzyskiwanie przy wielu małych plikach kosztem większego obciążenia sieci i procesora hosta HANA w tym samym momencie; zbyt wysoka wartość na słabszym łączu wydłuża odtwarzanie zamiast je przyspieszyć, bo wszystkie równoległe

strumienie zaczynają się wzajemnie dławić.

Pojęcie: `DownloadConcurrency`. Liczba równoległych połączeń HTTP, jakie agent otwiera do S3 przy pobieraniu pojedynczego pliku backupu — niezależna od `MaximumConcurrentFilesForRestore`, które mnoży liczbę plików, nie liczbę połączeń na plik. Dla dużych plików backupu (pełny backup wielogigabajtowej bazy) wyższa wartość `DownloadConcurrency` skraca czas pobrania jednego pliku, dzieląc go na równoległe fragmenty.

Przykładowa konfiguracja tych dwóch parametrów w pliku konfiguracyjnym agenta:

```
MaximumConcurrentFilesForRestore: 5
DownloadConcurrency: 10
```

Uważaj: oba parametry razem określają łączną liczbę jednoczesnych połączeń do S3 — przy `MaximumConcurrentFilesForRestore: 5` i `DownloadConcurrency: 10` agent może w szczycie otwierać do 50 połączeń naraz. Na hoście z ograniczoną przepustowością sieci warto obniżyć jeden z tych parametrów, zanim obniży się drugi, żeby zachować kontrolę nad tym, który wymiar równoległości faktycznie ogranicza.

Czas trwania kopii logów i „ostatnia strona" pliku logu

Pojęcie: kopia logów transakcyjnych (log backup). Ciągły, przyrostowy zrzut logu transakcyjnego HANA, uruchamiany automatycznie po zapełnieniu bieżącego segmentu logu — w odróżnieniu od backupu pełnego i różnicowego, log backup nie jest jednorazową operacją na żądanie, tylko procesem działającym w tle przez cały czas pracy bazy.

Przy kopiowaniu backupu logów do innej lokalizacji (np. przy replikacji katalogu backupu do ośrodka zapasowego) istotne jest to, że plik ostatniego, aktualnie otwartego segmentu logu (`log_backup_0_0_0_0` i podobne) rośnie i zmienia zawartość, dopóki segment nie zostanie zamknięty. Skopiowanie takiego pliku w trakcie jego zapisu daje kopię niekompletną — jeśli klucz obiektu w S3 został skopiowany wcześniej, a segment logu urósł już po tej kopii, trzeba skopiować nowszą wersję tego samego klucza ponownie, inaczej odtworzenie bazy z tej kopii zatrzyma się dokładnie na granicy, w której ostatnia skopiowana wersja się urwała.

Uważaj: przy każdym odświeżeniu kopii katalogu backupu do innej lokalizacji sprawdzamy, czy ostatni plik logu w źródle ma nowszą datę modyfikacji (albo inny rozmiar) niż jego kopia w celu — jeśli tak, kopiujemy go ponownie, zanim uznamy synchronizację za zakończoną.

Typowe komunikaty i co oznaczają

Komunikat	Znaczenie
`RECOVER DATA finished with error: recovery could not be completed`	proces odtwarzania zatrzymał się przed ukończeniem — właściwa przyczyna jest zwykle w komunikacie poniżej tej linii, nie w niej samej
`Error reading backup from BACKINT`	agent backint nie zwrócił oczekiwanych danych dla żądanego pliku backupu — najczęściej plik jest niekompletny albo nie istnieje pod oczekiwanym kluczem w S3
`Not all data could be written: Expected 4096 but transferred 0`	agent otworzył strumień do pliku backupu, ale nie odczytał z niego ani bajtu — typowy obraz sytuacji, w której kopiowany plik logu był w trakcie dopisywania (patrz sekcja wyżej) albo klucz w S3 wskazuje na starą, niepełną wersję obiektu
`Backint cannot find file`	agent nie znalazł w katalogu backupu (w S3, pod wskazanym prefiksem) pliku, którego żąda HANA na podstawie katalogu backupu bazy — zwykle plik został przeniesiony, usunięty przez regułę cyklu życia opisaną na stronie S3: kubełki, obiekty, uprawnienia, albo nigdy nie dotarł do docelowej lokalizacji

Diagnozowanie tych komunikatów zaczynamy zawsze od sprawdzenia, czy plik backupu, o który pyta HANA, faktycznie istnieje pod oczekiwanym kluczem w S3 i czy jego rozmiar odpowiada oczekiwanemu — dopiero potem sprawdzamy uprawnienia i limity równoległości agenta.

Odtwarzanie bazy z backupu w S3

Proces odtwarzania (recovery) w studiu administracyjnym HANA pyta kolejno o lokalizację backupu danych, backupu logów i katalogu backupu — przy backincie wskazującym S3 każda z tych lokalizacji jest ścieżką wewnątrz kubełka (prefiksem), a nie ścieżką w lokalnym systemie plików. Zanim proces odtwarzania cokolwiek zmieni, HANA opcjonalnie sprawdza dostępność wszystkich wymaganych backupów delta i logów — jeśli którychś brakuje, odtwarzanie zatrzymuje się przed wprowadzeniem jakiejkolwiek zmiany, zamiast ryzykować częściowe odtworzenie. Warto zostawiać tę weryfikację włączoną mimo wydłużenia czasu przygotowania — powtórzenie całego odtwarzania od zera po odkryciu braku w jego trakcie kosztuje zdecydowanie więcej czasu niż samo sprawdzenie na starcie.

Częste błędy

Kopiowanie katalogu backupu do ośrodka zapasowego bez ponownego sprawdzenia ostatniego segmentu logu. Jednorazowe skopiowanie całego katalogu backupu wygląda na kompletne, ale jeśli ostatni segment logu wciąż rósł w trakcie kopiowania, odtworzenie z tej kopii urwie się dokładnie na tym segmencie — konieczne jest odświeżenie przynajmniej ostatniego pliku przed uznaniem kopii za gotową do użycia.

Podniesienie `MaximumConcurrentFilesForRestore` bez sprawdzenia przepustowości łącza. Wyższa wartość nie przyspiesza odtwarzania automatycznie — na wolniejszym łączu równoległe strumienie konkurują o to samo pasmo i całkowity czas odtworzenia może się wydłużyć zamiast skrócić.

Traktowanie komunikatu `Backint cannot find file` jako problemu z samą HANA. Przyczyna leży niemal zawsze po stronie magazynu backupu (usunięty obiekt, zła nazwa prefiksu, wygasła reguła cyklu życia) — HANA tylko przekazuje błąd zwrócony przez agenta backint.

□□ Ćwiczenia

> Zadania sprawdzające, bez dostępu do systemu SAP HANA — wystarczy zrozumienie mechanizmu.

1. Wyjaśniamy, dlaczego HANA „nie wie", że jej backup trafia do S3, mimo że faktycznie tam ląduje.
2. Opisujemy różnicę między `MaximumConcurrentFilesForRestore` a `DownloadConcurrency` i podajemy, w jakiej sytuacji obniżenie jednego z nich pomaga bardziej niż obniżenie drugiego.
3. Odtwarzanie bazy zatrzymuje się z komunikatem `Not all data could be written: Expected 4096 but transferred 0`. Wymieniamy dwie najbardziej prawdopodobne przyczyny i sposób ich rozróżnienia.
4. Uzasadniamy, dlaczego warto zostawić włączoną weryfikację dostępności backupów delta i logów przed rozpoczęciem odtwarzania, mimo że wydłuża to czas przygotowania procesu.

Powiązane: Backup pełny, różnicowy, przyrostowy, logi · S3: kubełki, obiekty, uprawnienia · Architektura SAP HANA: instancja, numer, nazwa SID, procesy

Źródła: dokumentacja SAP HANA, interfejs Backint · dokumentacja agenta backint dla Amazon S3