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:

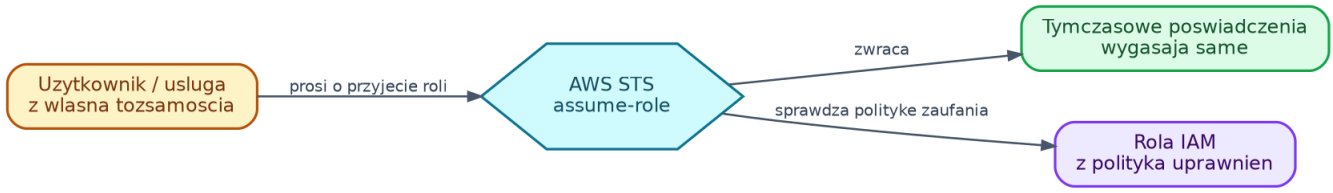
```
{
    "UserId": "AIDACKCEVSQ6C2EXAMPLE",
    "Account": "123456789012",
    "Arn": "arn:aws:iam::123456789012:user/nazwa_usera"
}
```

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

---

Revision #37
Created 2026-09-25 08:26:19 UTC by Tomasz Konieczny
Updated 2026-10-26 14:16:21 UTC by Tomasz Konieczny