# 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.

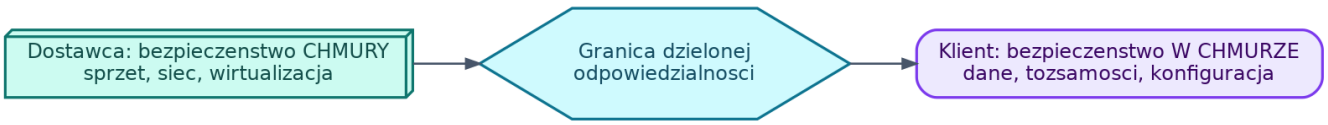
Rozróżniamy trzy podstawowe modele:

- **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

---

Revision #37
Created 2026-09-25 08:26:17 UTC by Tomasz Konieczny
Updated 2026-10-26 14:16:20 UTC by Tomasz Konieczny