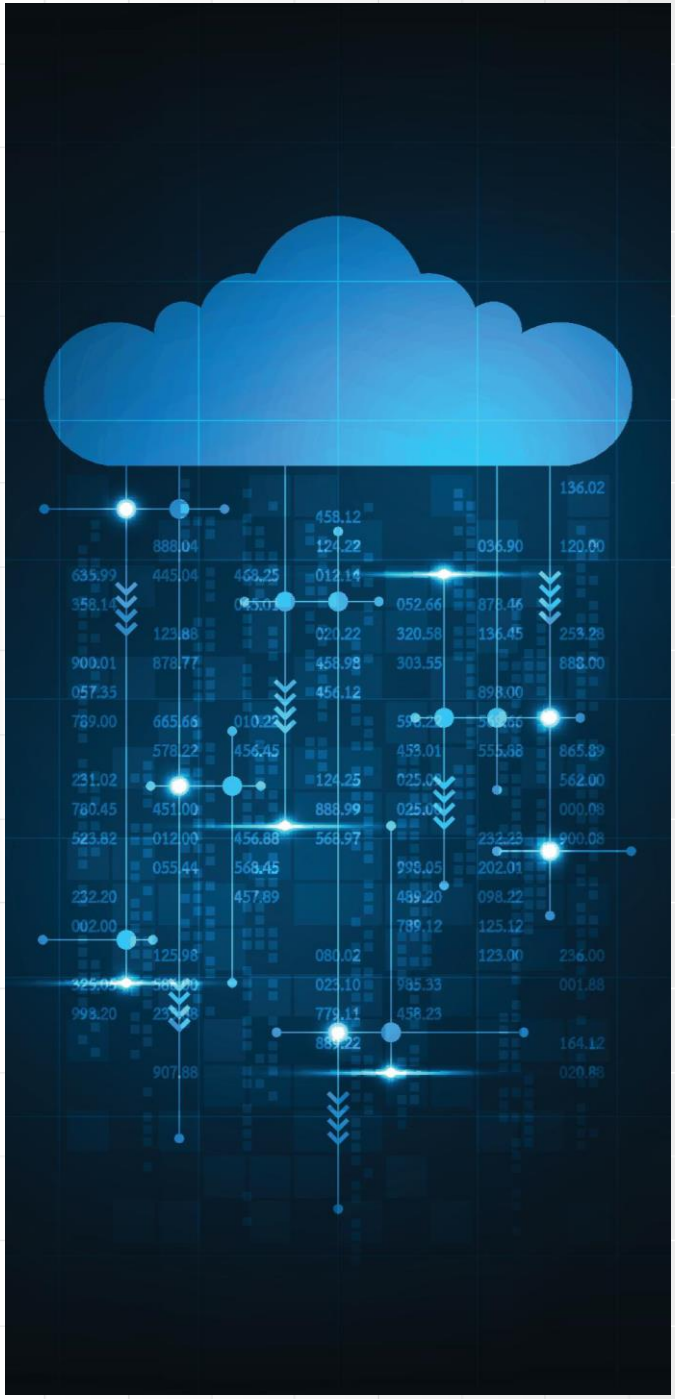


Programowanie w chmurze

Rafał Palak

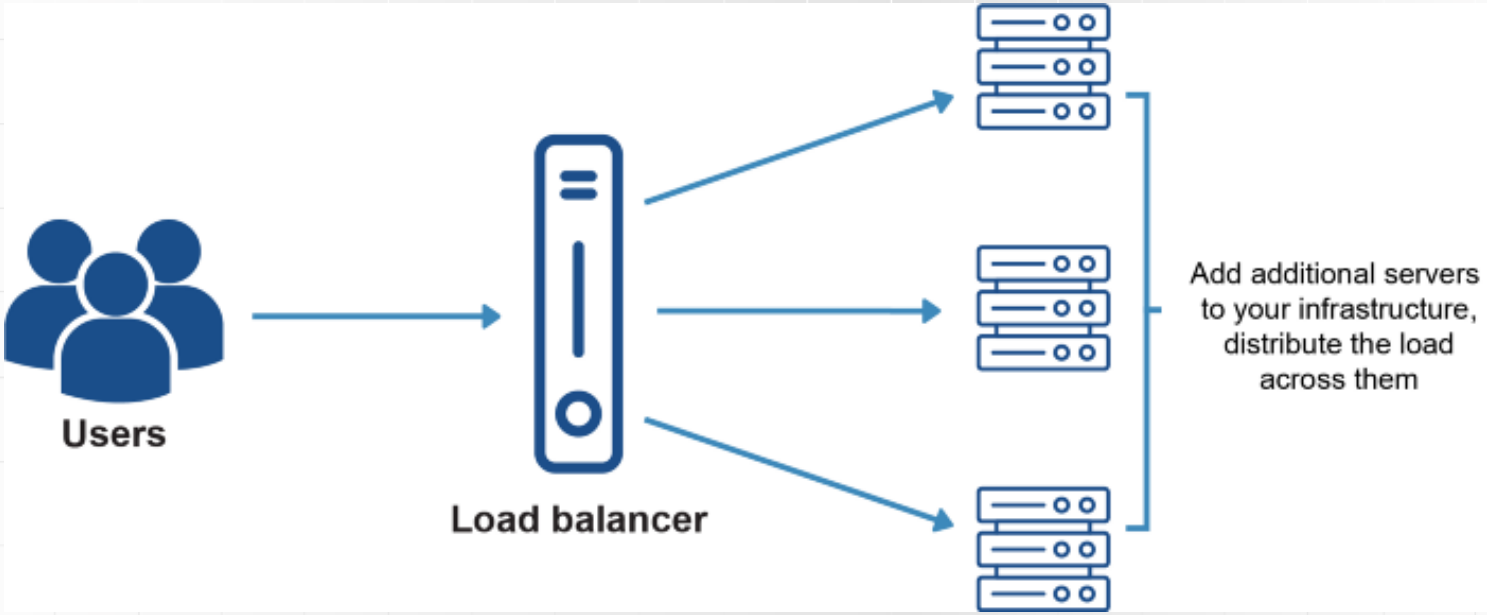
Politechnika Wrocławska

Wrocław
University
of Science
and Technology

Zasady projektowania skalowalnych architektur

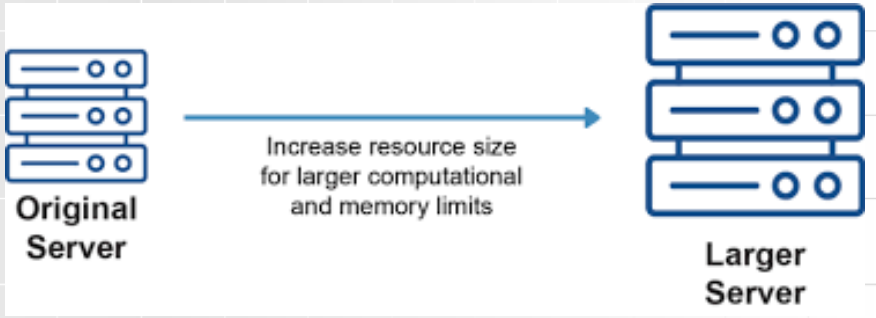
Skalowalność

- Skalowalność to zdolność systemu do obsługi rosnącego obciążenia przez dodawanie zasobów.



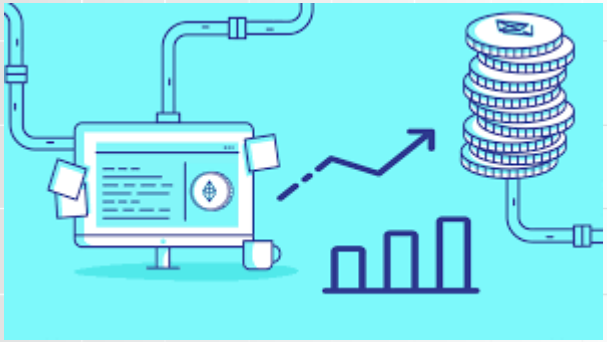
Skalowalność - rodzaje

- Skalowalność Pozioma (Horizontal Scaling
  - Lepsza odporność na awarie (fault tolerance).
  - Możliwość niemal nieograniczonego skalowania.
  - Złożoność zarządzania wieloma instancjami.
  - Potrzeba mechanizmów równoważenia obciążenia (load balancing).
- Skalowalność Pionowa (Vertical Scaling)
  - Prostsze zarządzanie w porównaniu ze skalowalnością poziomą.
  - Brak potrzeby modyfikacji aplikacji pod kątem wielu instancji.
  - Ograniczona maksymalna pojemność jednego serwera.
  - Ryzyko awarii pojedynczego punktu (single point of failure).



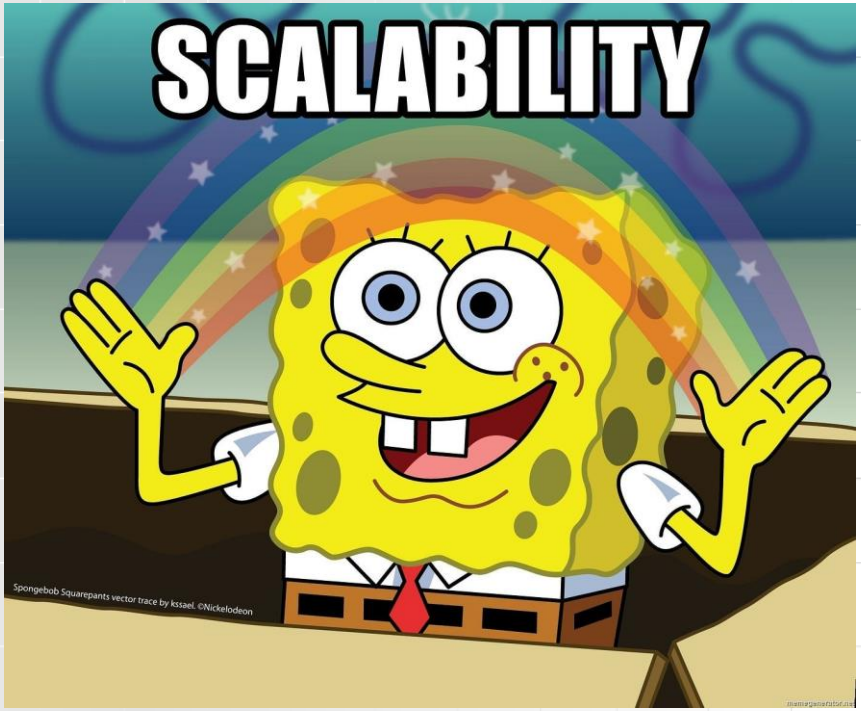
Skalowalność – dlaczego?

- Zdolność do obsługi większej liczby użytkowników bez pogorszenia wydajności.
- Możliwość dynamicznego dostosowywania się do zmieniających się wymagań.
- System może nadal działać mimo awarii pojedynczych komponentów (szczególnie w przypadku skalowalności poziomej).
- Płacenie tylko za zasoby, które są aktualnie potrzebne.



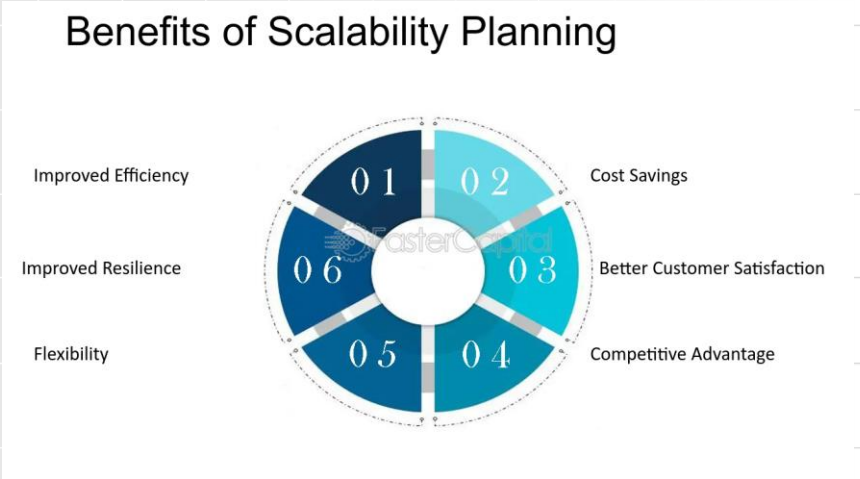
Skalowalność - wyzwania

- Zarządzanie wieloma instancjami może być skomplikowane.
- Koszty związane z dodatkowymi zasobami i infrastrukturą.
- Potrzeba odpowiednich mechanizmów do równomiernego rozkładania ruchu.



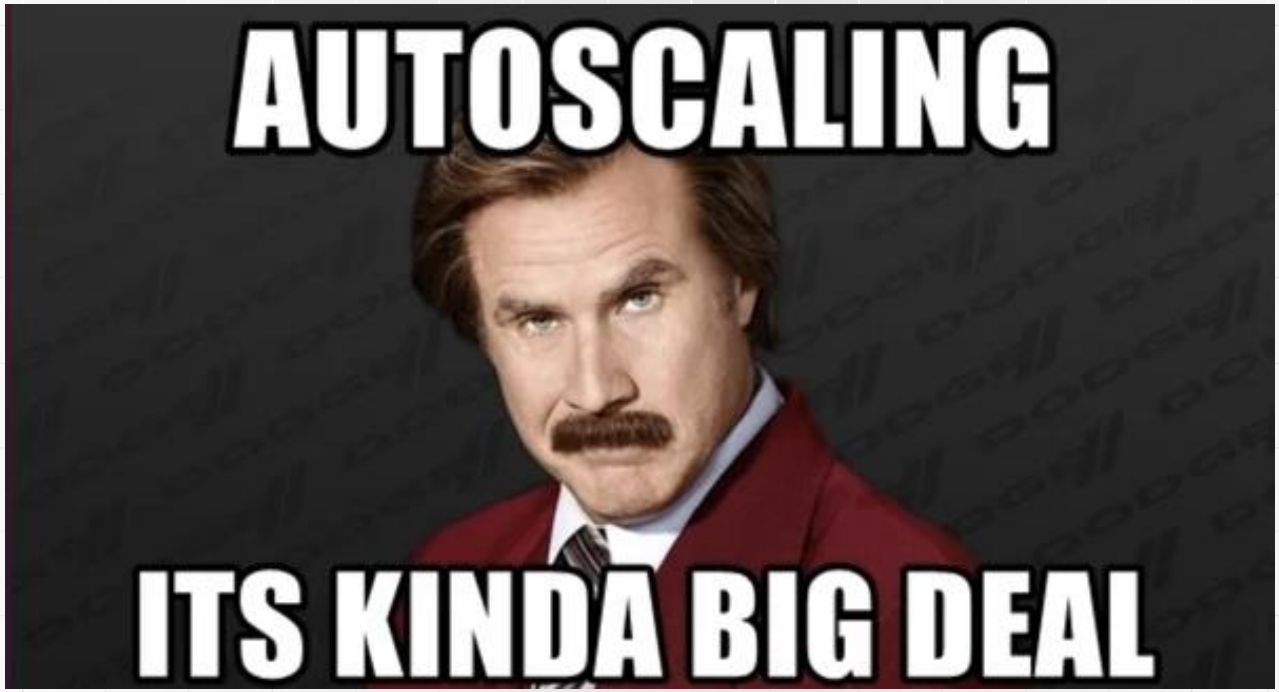
Skalowalność – co wybrać?

- Czy aplikacja ma zmienne czy stałe obciążenie?
- Czy aplikacja wymaga wysokiej wydajności w pojedynczej instancji?
- Czy aplikacja jest bezstanowa?
- Jaki jest budżet na infrastrukturę?
- Czy aplikacja wymaga wysokiej dostępności i odporności na awarie?
- Jak szybko trzeba dostosować się do zmieniających się warunków rynkowych lub liczby użytkowników?
- Jakie zasoby są dostępne?
- Czy technologie i frameworki używane w aplikacji wspierają jeden typ skalowalności lepiej niż inny?



Elastyczność

- Elastyczność odnosi się do zdolności systemu do automatycznego dostosowywania zasobów w odpowiedzi na zmieniające się obciążenie.

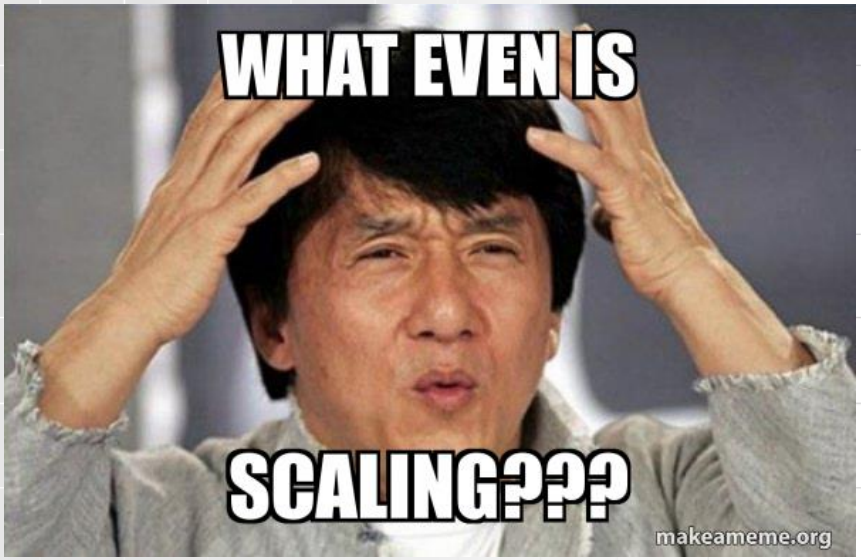


Elastyczność - zalety

- Automatyczne skalowanie zasobów w górę lub w dół w zależności od obciążenia.
- Minimalizacja kosztów poprzez dynamiczne dostosowywanie zasobów do aktualnych potrzeb.
- Zdolność do natychmiastowego reagowania na zmieniające się wymagania biznesowe i technologiczne.

Elastyczność – przykłady elastyczności

- Automatyczne dodawanie lub usuwanie instancji EC2 na podstawie ustalonych zasad.
- Automatyczne skalowanie funkcji bezserwerowych w odpowiedzi na liczbę żądań.
- Automatyczne dostosowywanie zasobów bazy danych w zależności od obciążenia.



Elastyczność – wyzwania

- Konieczność prawidłowego skonfigurowania zasad skalowania.
- Niezbędność ciągłego monitorowania i zarządzania zasobami.
- Potencjalne zwiększenie kosztów w przypadku niewłaściwej konfiguracji automatycznego skalowania.

Elastyczność – co wybrać?

- Czy aplikacja wymaga szybkiego skalowania w górę i w dół?
- Czy aplikacja musi obsługiwać różnorodne obciążenia w różnych porach dnia?
- Jakie są wymagania dotyczące niezawodności i odporności na awarie?
- Jakie są potrzeby dotyczące automatyzacji i zarządzania infrastrukturą?
- Czy aplikacja wymaga elastycznego zarządzania danymi?
- Jakie są wymagania dotyczące integracji z innymi systemami i usługami?

Wrocław
University
of Science
and Technology

Zasady projektowania skalowalnych architektur

Projektowanie skalowalnych architektur

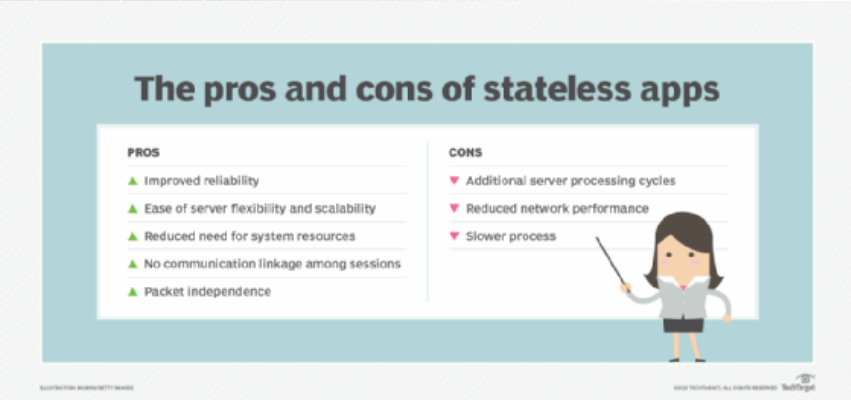
- Aplikacje bezstanowe
- Asynchroniczność

Aplikacje bezstanowe

- Oznacza, że serwer nie przechowuje żadnych danych dotyczących stanu aplikacji między różnymi żądaniami od tego samego klienta.
- Każde żądanie od klienta jest niezależne i nie wpływa na inne.

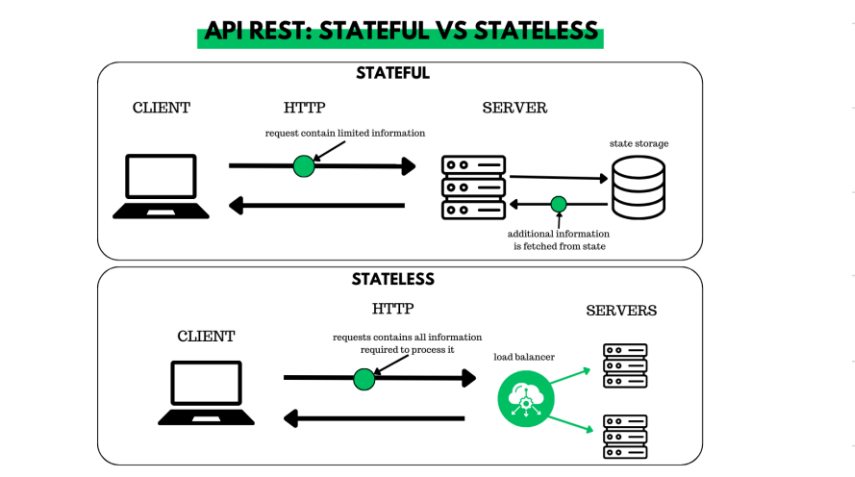
Aplikacje bezstanowe - zalety

- **Łatwiejsze skalowanie poziome** - nowe instancje serwerów mogą obsługiwać każde żądanie bez potrzeby synchronizacji stanu.
- **Możliwość dynamicznego dodawania lub usuwania instancji** serwerów bez wpływu na sesje użytkowników.
- **Mniejsze ryzyko awarii**, ponieważ brak konieczności replikacji stanu między serwerami.



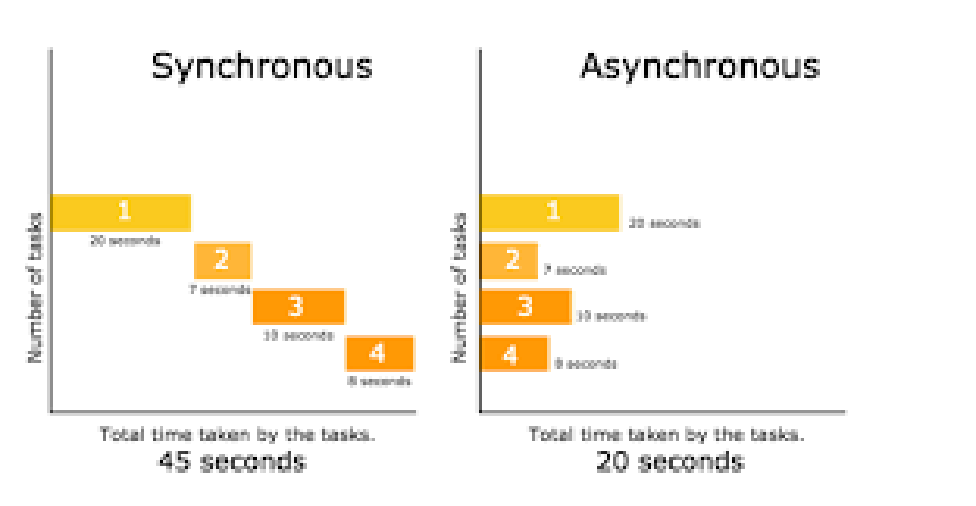
Aplikacje bezstanowe – jak osiągnąć?

- **Przechowywanie stanu po stronie klienta** - Używanie tokenów JWT (JSON Web Token) do przechowywania stanu na kliencie.
- **Przechowywanie stanu w bazie danych** - Trzymanie sesji użytkowników w szybkich bazach danych, takich jak Redis lub DynamoDB.
- **Przechowywanie stanu w pamięci podręcznej** - Używanie systemów cache'owania, jak Amazon ElastiCache.



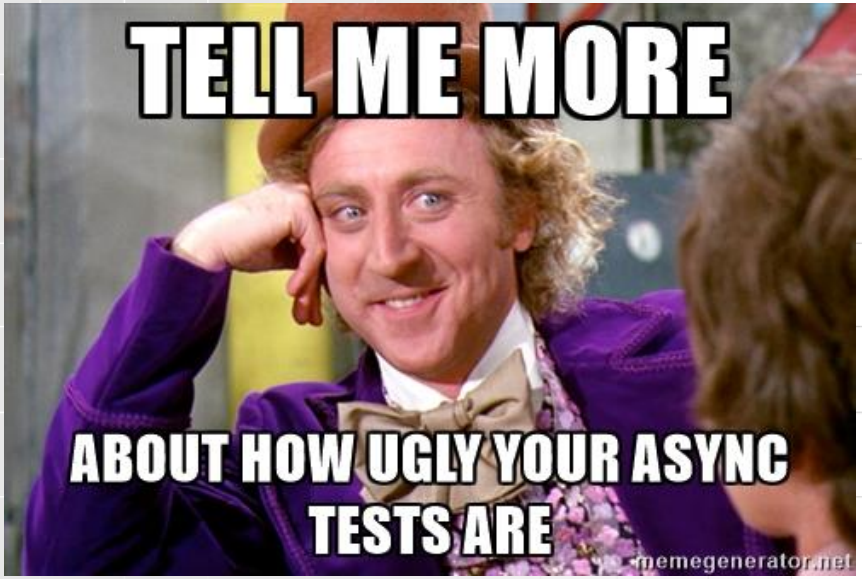
Asynchroniczność

- Asynchroniczność umożliwia systemom wykonywanie zadań równocześnie, bez oczekiwania na zakończenie innych operacji.
- Kluczowy element do osiągnięcia skalowalności i elastyczności w systemach rozproszonych.



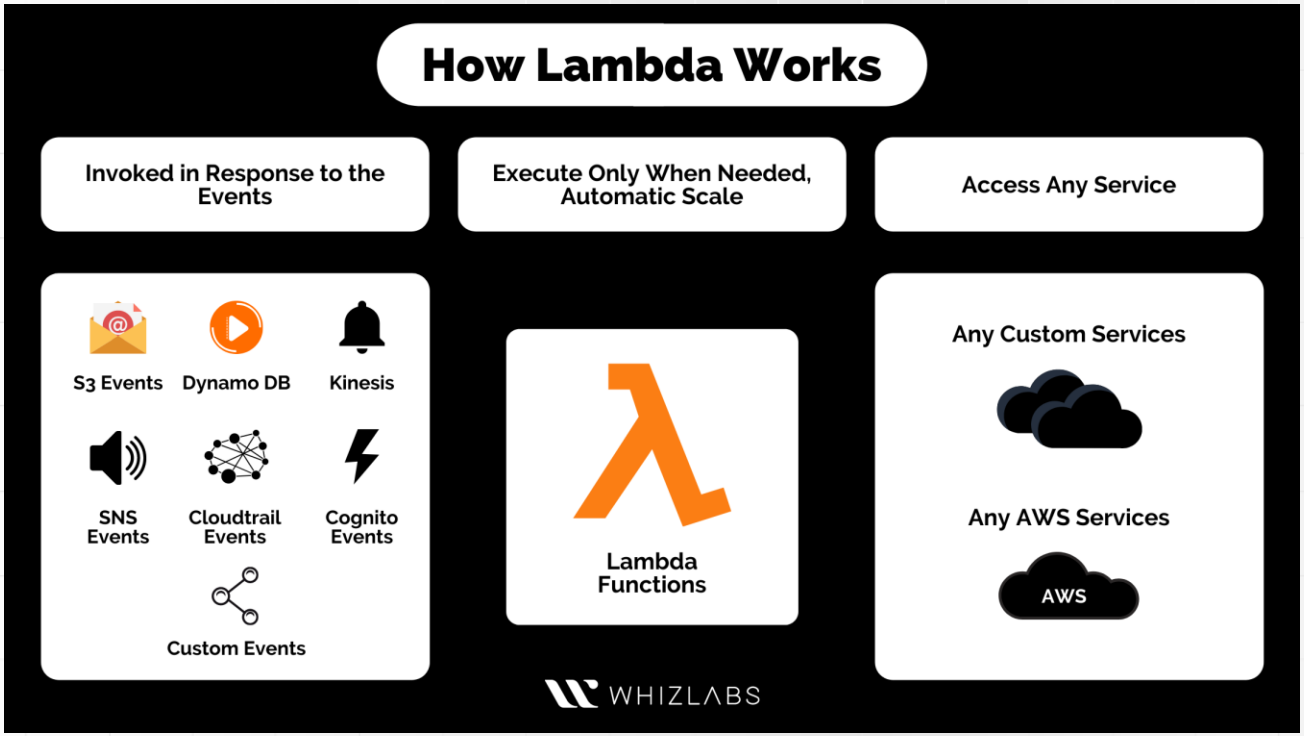
Asynchroniczność - zalety

- **Poprawa wydajności -** Możliwość przetwarzania wielu zadań równocześnie.
- **Łatwiejsze skalowanie** aplikacji.
- **Lepsza obsługa błędów -** Możliwość izolowania i zarządzania błędami w poszczególnych komponentach.
- **Responsywność -** Szybsza reakcja na zdarzenia użytkowników.



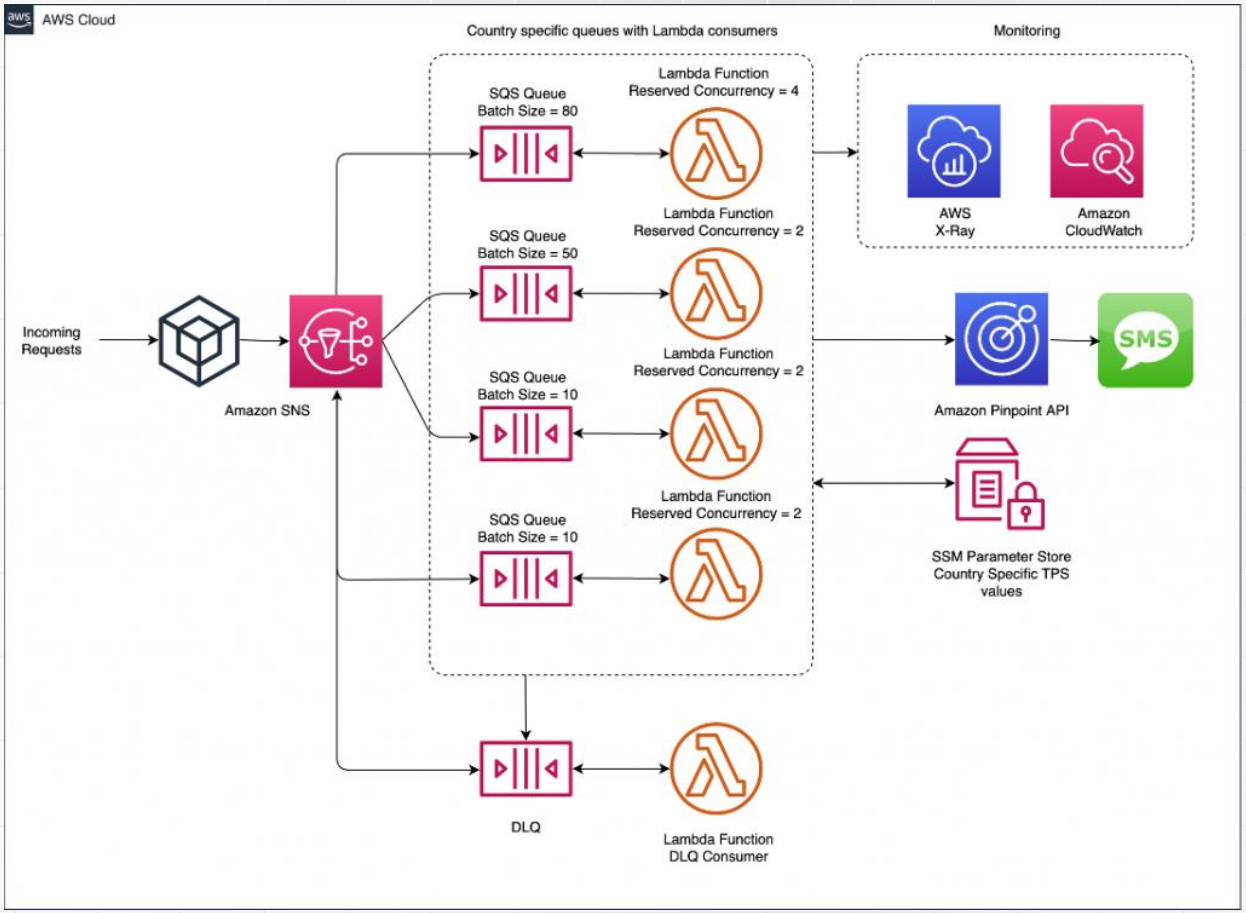
Asynchroniczność - AWS Lambda

- Funkcje bezserwerowe, które mogą być wyzwalane przez zdarzenia.



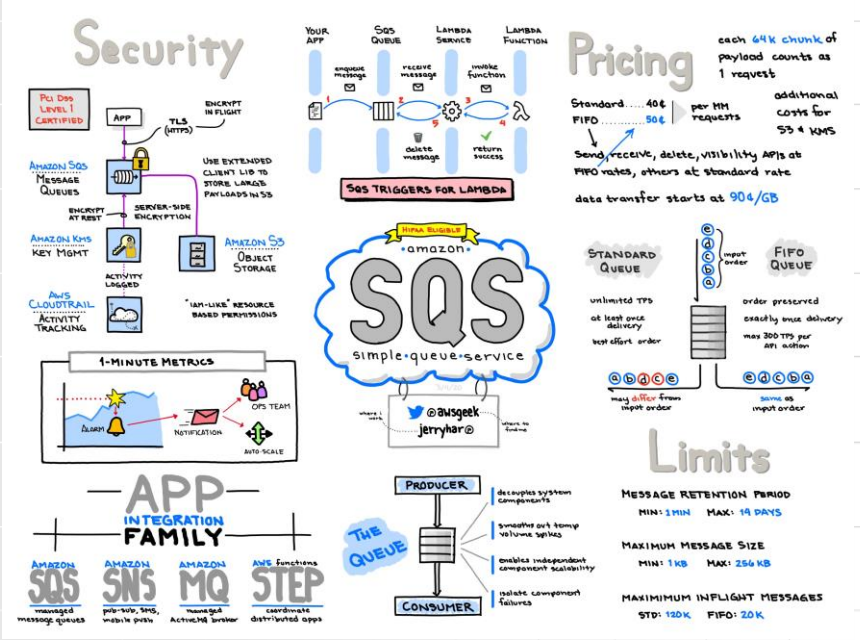
Asynchroniczność - Amazon SQS

- Kolejki wiadomości do buforowania i przesyłania komunikatów między komponentami.



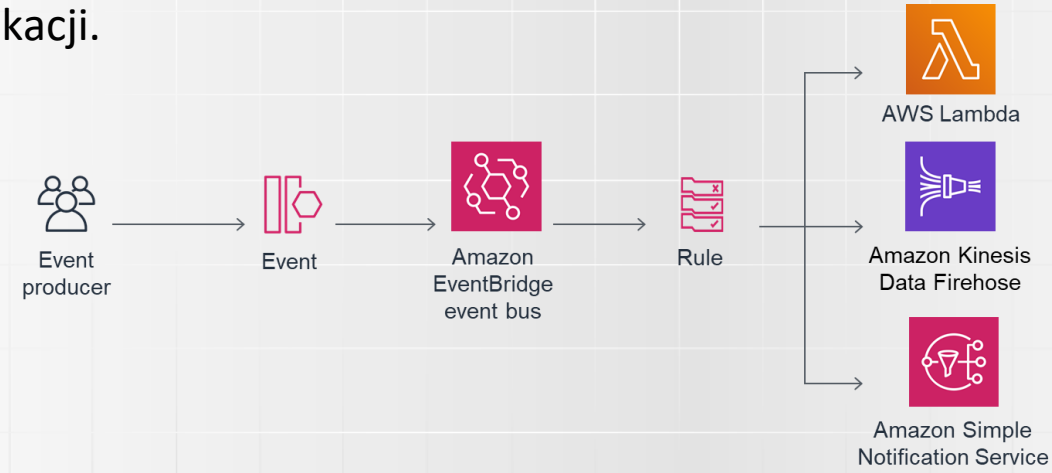
Asynchroniczność - Amazon SQS

- SQS umożliwia rozdzielenie komponentów aplikacji, co zwiększa skalowalność i niezawodność systemu.
- **Skalowalność** - Automatyczne skalowanie umożliwia obsługę nieograniczonej liczby wiadomości.
- **Trwałość** - Wiadomości są przechowywane w wielu lokalizacjach, co zapewnia ich trwałość i dostępność.
- **Czas Widoczności (Visibility Timeout)** - Mechanizm, który pozwala na czasowe ukrycie wiadomości przed innymi konsumentami podczas jej przetwarzania.
- **Kolejka Dead-Letter**- Mechanizm pozwalający na przechwytywanie i analizowanie wiadomości, które nie mogły zostać przetworzone.



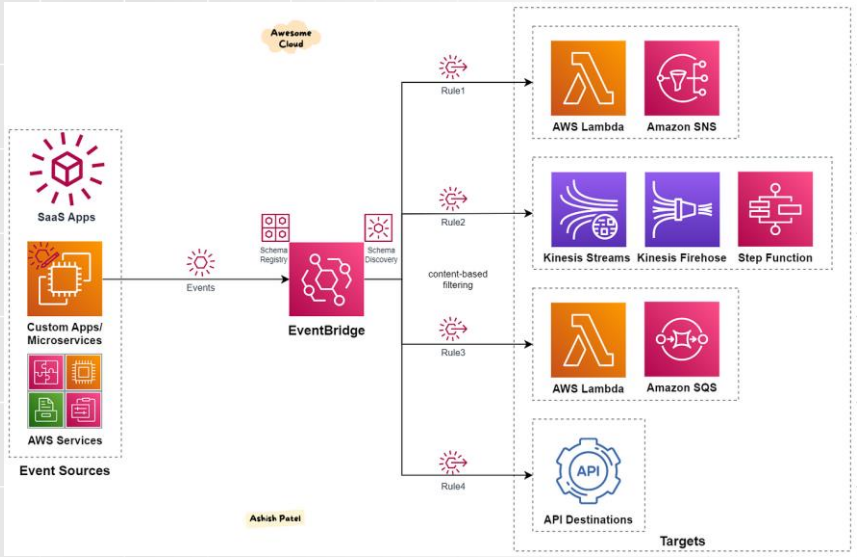
Asynchroniczność - Amazon EventBridge

- Usługa zarządzania zdarzeniami, pozwalająca na tworzenie reguł wyzwalających różne działania na nie w czasie rzeczywistym.
- Umożliwia budowanie aplikacji zorientowanych na zdarzenia poprzez łączenie różnych usług AWS, aplikacji SaaS (Software-as-a-Service) oraz własnych aplikacji.



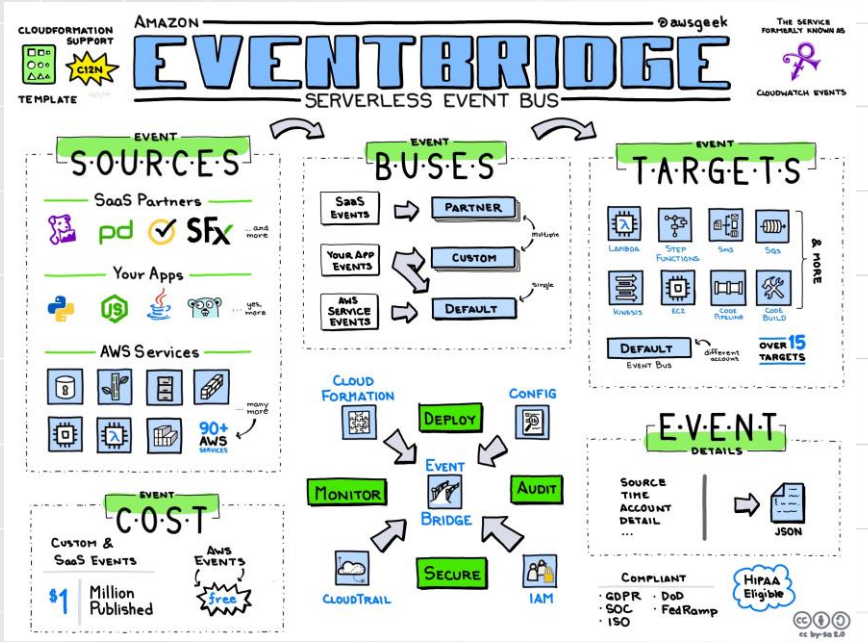
Amazon EventBridge - Podstawowe Koncepcje

- Zdarzenie (Event): Informacja o zmianie stanu w systemie, np. zakończenie przetwarzania pliku, zmiana statusu zamówienia itp.
- Bus Zdarzeń (Event Bus): Kanał, przez który przepływają zdarzenia, mogący odbierać zdarzenia z różnych źródeł i dostarczać je do różnych celów.
- Źródła Zdarzeń (Event Sources): Usługi AWS, aplikacje SaaS, a także własne aplikacje, które generują zdarzenia.
- Cele Zdarzeń (Event Targets): Usługi lub aplikacje, które reagują na zdarzenia, np. AWS Lambda, Step Functions, SNS, SQS.



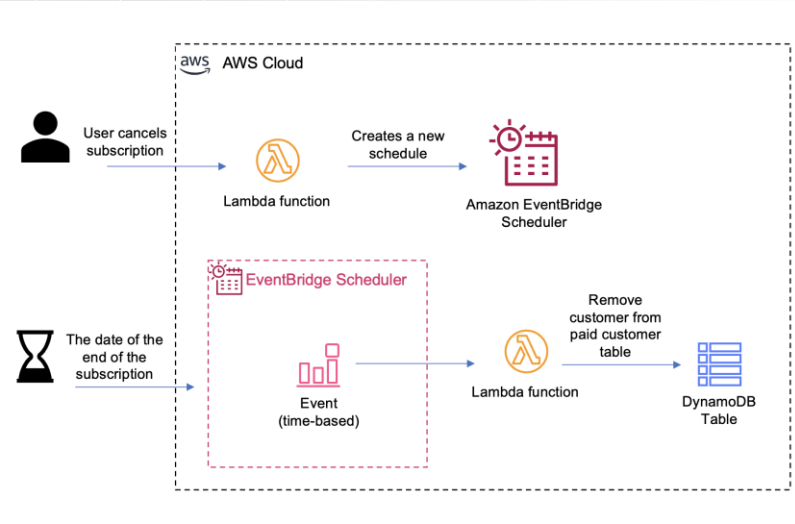
Amazon EventBridge - Funkcjonalności i Zalety

- Integracja z AWS: EventBridge integruje się bezpośrednio z wieloma usługami AWS, co ułatwia budowanie kompleksowych rozwiązań.
- Obsługa Aplikacji SaaS: Możliwość odbierania zdarzeń z popularnych aplikacji SaaS takich jak Zendesk, Datadog, czy Shopify.
- Elastyczność: Możliwość definiowania reguł, które określają, jakie zdarzenia są przekazywane do jakich celów.
- Bezpieczeństwo: Kontrola dostępu za pomocą AWS IAM oraz szyfrowanie danych w ruchu i w spoczynku.
- Monitoring i Analiza: Integracja z Amazon CloudWatch umożliwia monitorowanie zdarzeń i analizę logów.



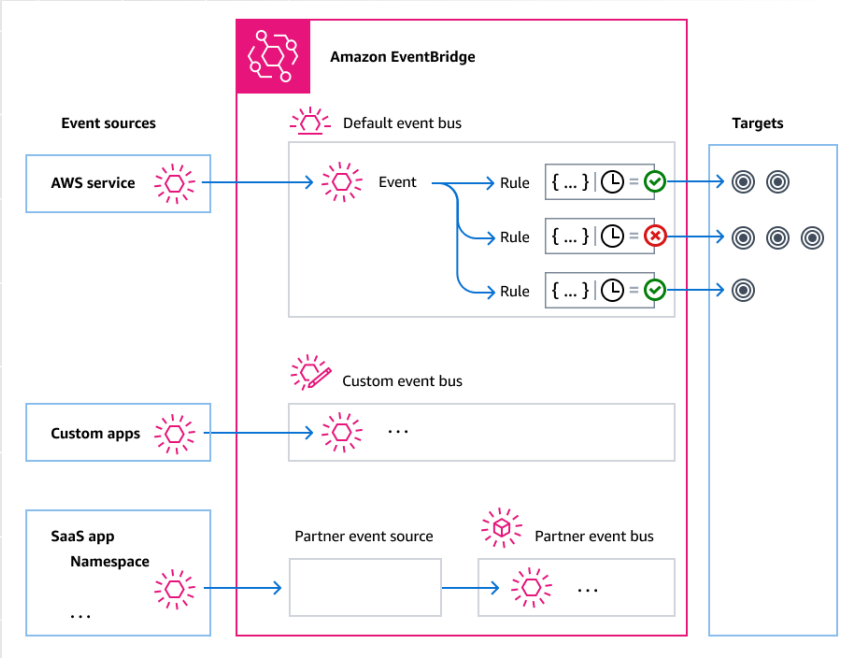
Amazon EventBridge - Architektura i Działanie

- Struktura Zdarzenia: Każde zdarzenie ma standardowy format JSON, zawierający informacje takie jak źródło zdarzenia, czas, dane specyficzne dla danego zdarzenia.
- Przepływ Zdarzeń: Proces od wygenerowania zdarzenia przez źródło, poprzez przesłanie go do busa zdarzeń, aż po dostarczenie do celu.
- Reguły Zdarzeń: Definiowanie reguł, które określają, jakie zdarzenia są kierowane do jakich celów. Reguły mogą filtrować zdarzenia na podstawie ich zawartości.



Amazon EventBridge - Przykłady Zastosowań

- Automatyzacja Operacji IT: Reagowanie na zmiany stanu zasobów AWS, takie jak zakończenie instancji EC2, tworzenie nowych plików w S3 itp.
- Przetwarzanie Danych: Automatyczne uruchamianie przetwarzania danych w AWS Lambda, gdy nowe dane są dostępne.
- Integracje Aplikacji SaaS: Synchronizacja danych między aplikacjami SaaS a zasobami AWS.



Wrocław
University
of Science
and Technology

Dziękuję za uwagę