

## ADR-044 Korzystanie z architektury rozproszonej sterowanej zdarzeniami

### Status

Zatwierdzony, 04.10.2023

Zastępuje [ADR-031 Używanie funkcji bezserwerowych]

### Kontekst

System Polyglot Media jest obecnie systemem rozproszonym, składającym się głównie z funkcji serverless. Przejście z architektury monolitycznej na rozproszoną architekturę serverless miało na celu rozwiązanie problemów z responsywnością systemu w trakcie działania i długim czasem oczekiwania na przygotowanie i wdrożenie w środowisku produkcyjnym zarówno nowych funkcjonalności, jak i poprawek.

Architektura serverless nie rozwiązała problemów z responsywnością w wymaganym stopniu ani nie zapewniła oczekiwanej łatwości utrzymania z powodu bardzo wysokiego poziomu powiązań pomiędzy poszczególnymi funkcjami.

Należy znaleźć rozwiązanie problemów z szybkością reakcji i czasem wprowadzania produktu na rynek.

### Kryteria oceny

Zobacz [ADR-002 Wybór cech architektury].

- **Responsywność:** Klienci skarżyli się na problemy z responsywnością systemu i należy się tym zająć. Ta cecha jest uważana za najważniejsze kryterium.
- **Łatwość utrzymania:** Funkcje serverless skróciły czas wprowadzania na rynek poprawek błędów i nowych funkcjonalności, ale nie w wymaganym stopniu.
- **Łatwość wdrażania:** Oprócz łatwości konserwacji łatwość wdrażania jest ważna dla skrócenia czasu wprowadzania na rynek poprawek błędów i nowych funkcji.
- **Skalowalność:** Większość skarg klientów z zakresu czasu reakcji dotyczyła szczytowych okresów największego obciążenia, więc system musi obsługiwać maksymalną przewidywaną liczbę użytkowników bez wpływu na wydajność działania.

### Opcje

#### 1. Mikrouslugi

Kryteria	Wynik	Uzasadnienie
Responsywność	★★★★☆ 3/5	[Nie jest wydajny z natury], ale można wprowadzić optymalizacje w wąskich gardłach, np. poprzez skalowanie.
Łatwość utrzymania	★★★★☆ 3/5	Zależności mogą stanowić problem, konieczność utrzymania wielu magazynów danych.
Łatwość wdrażania	★★★★★ 5/5	Wdrażane jest tylko to, co się zmieniło.
Skalowalność	★★★★★ 5/5	Można skalować wybrane usługi.
	Razem: 16/20	<b>Inne kompromisy:</b> <ul style="list-style-type: none"><li>▪ Konieczność dzielenia danych na [jeden magazyn danych na usługę].</li><li>▪ Koszty budowy są zazwyczaj wysokie.</li><li>▪ Złożoność i trudność w tworzeniu przepływów pracy.</li></ul>

#### 2. Rozwiązanie oparte na usługach

Kryteria	Wynik	Uzasadnienie
Responsywność	★★★★☆ 3/5	[Nie jest wydajny z natury], ale można wprowadzić optymalizacje w wąskich gardłach, np. poprzez skalowanie.
Łatwość utrzymania	★★★★☆ 4/5	Mniej magazynów danych do utrzymania niż w przypadku wykorzystania mikrouslug.

Łatwość wdrażania	★★★★☆ 4/5	Wdrażane jest tylko to, co zostało zmienione; trzeba upewnić się, że zmiany we współdzielonych magazynach danych nie mają wpływu na inne usługi.
Skalowalność	★★★★☆ 3/5	Skalowanie poszczególnych usług, trudniejsze skalowanie współdzielonych magazynów danych, chyba że przy wykorzystaniu [buforowania].
	Razem: 14/20	<b>Inne kompromisy:</b> <ul style="list-style-type: none"> <li>▪ Rozsądnie złożone i trudne do stworzenia przepływy pracy.</li> <li>▪ Nie tak ewolucyjne jak mikrousługi.</li> </ul>

### 3. Rozwiązanie sterowane zdarzeniami

Kryteria	Wynik	Uzasadnienie
Responsywność	★★★★★ 5/5	Ogólnie rozwiązanie typu [odpal i zapomnij], przetwarzanie zdarzeń może być skalowane lub zoptymalizowane.
Łatwość utrzymania	★★★★☆ 4/5	Musi zarządzać przetwarzaniem zdarzeń, a także usługami i magazynami danych, zdarzenia oddzielają usługi do zarządzania przez poszczególne zespoły.
Łatwość wdrażania	★★★☆☆ 3/5	Zarządzanie wdrażaniem i zmianami przetwarzania zdarzeń oraz usługami.
Skalowalność	★★★★★ 5/5	Usługi i przetwarzanie wiadomości mogą być skalowane.
	Razem: 17/20	<b>Inne kompromisy:</b> <ul style="list-style-type: none"> <li>▪ [Testowanie integracyjne może być trudniejsze].</li> </ul>

## Decyzja

Zastosujemy architekturę sterowaną zdarzeniami, opartą na całkowitym wyniku w odniesieniu do kryteriów decyzyjnych oraz na fakcie, że opcje 1 i 2 nie uzyskały dobrych wyników w najważniejszym kryterium — responsywności.

## Skutki

### Pozytywy

- Zyskujemy ogólnie wyższą responsywność, łatwość utrzymania i skalowalność niż te, które zapewniają obecnie stosowane funkcje serverless.
- Przetwarzanie zdarzeń jest skalowalne poprzez uruchamianie dodatkowych instancji usług przetwarzających zdarzenia pobierane z kolejki.
- Przetwarzanie w ramach usług jest skalowalne poprzez uruchamianie dodatkowych instancji danej usługi w celu zaspokojenia zapotrzebowania na wykonywane przez nią działania.

### Negatywy

- Może wystąpić konieczność zdobycia przez zespoły lub osoby nowych umiejętności albo opanowania nowych technologii (takich jak zarządzanie kolejkami zdarzeń).
- Zarządzanie zdarzeniami (takimi jak kolejki) dodatkowo komplikuje proces programowania i wdrażania.
- Testy integracyjne i DevOps będą musiały zostać ponownie zweryfikowane.

## Konsultacje

- *Wladek*: Ani mikrousługi, ani usługi nie są [z natury] wydajne, więc prawdopodobnie nie zwiększą szybkości reakcji tak bardzo, jak byśmy chcieli.
- *Natalia*: Mam doświadczenie w architekturze sterowanej zdarzeniami, wykorzystującej kolejki. Monitorowaliśmy długość kolejek i w razie potrzeby uruchamialiśmy dodatkowe instancje usług do przetwarzania długich kolejek.
- *Lidia*: Kolejki i kanały mogą być wykorzystywane do nadawania priorytetów przetwarzaniu zdarzeń, na przykład za pomocą wzorca [kolejek priorytetowych].
- *Marek*: Ze względu na skargi klientów szybkość reakcji musi być najwyższym priorytetem.

- *Lidia*. Czysta architektura oparta na mikrouslugach wymagałaby, aby poszczególne usługi nie współdzieliły magazynów danych. Podzielenie naszych danych w taki sposób wymagałoby dużego wysiłku.