

# Analiza mikroservisów

<b>Streszczenie</b>	<b>3</b>
Cel	3
Sposób realizacji	3
Konkluzja	3
<b>Analiza</b>	<b>3</b>
Korzyści wykorzystania mikroserwisów	3
Zmapowane ryzyka	3
<b>Analiza argumentów</b>	<b>6</b>
<b>Konsultacje z NASK</b>	<b>7</b>
Użycie pamięci operacyjnej	7

# Streszczenie

## Cel

Podjęcie decyzji, czy platforma aplikacje.gov.pl będzie budowana w oparciu o wzorzec mikroserwisów (<http://microservices.io/>).

## Sposób realizacji

Analiza korzyści i zagrożeń wynikających z podjęcia decyzji o zbudowaniu platformy zgodnie ze wzorcem mikroserwisów. Analiza wagi poszczególnych argumentów. Na podstawie tej analizy podjęcie decyzji w sprawie użycia wzorca mikroserwisów.

## Konkluzja

Wzorzec mikroserwisów bardziej niż architektura monolityczna odpowiada potrzebom i założeniom aplikacji.gov.pl. Aby podjąć decyzję wykorzystania tego wzorca konieczne były konsultacje z zespołem odpowiedzialnym za infrastrukturę chmury. W wyniku konsultacji nie stwierdzono przeciwwskazań do zastosowania wzorca mikrousług. W związku z tym podjęto decyzję o budowie platformy w oparciu o wzorzec mikrousług.

## Analiza

### Korzyści wykorzystania mikroserwisów

- Różne części systemu mogą być tworzone w różnych technologiach.
- Zarządzanie zmianą i deploymentem jest łatwiejsze, niż w przypadku architektury monolitycznej.
- Łatwa wymiana konkretnej usługi na inną, w innej technologii, implementującą to samo API.
- Zarządzanie separacją aplikacji jest łatwiejsze, niż w przypadku architektury monolitycznej, co może wpływać pozytywnie na poziom bezpieczeństwa rozwiązania.
- Rozpraszanie aplikacji pomiędzy wiele maszyn jest łatwiejsze, niż w przypadku architektury monolitycznej.

### Zmapowane ryzyka

W dyskusjach środowiskowych o stosowaniu mikroserwisów pojawiają się argumenty przeciwko temu rozwiązaniu<sup>1</sup>. Przygotowując analizę zebraliśmy je w jednym miejscu punktując zasadność krytyki. Podzieliliśmy ryzyka na takie, które zgodnie z przytoczonymi

---

<sup>1</sup> Niektóre z nich zostały wyrażone we wpisie na blogu: <http://www.ictshop.pl/czym-sa-mikrouslugi/>

przez nas argumentami są niezasadne lub nie wpływają w istotny sposób na poziom trudności systemu i takie, które istotnie występują i muszą być wzięte pod uwagę w podejmowaniu decyzji o wykorzystaniu mikroserwisów

L.p.	Ryzyko	Kontrargumenty
1	Różne części systemu mogą być tworzone w różnych technologiach, co utrudnia zrozumienie całego kodu na szczegółowym poziomie.	Zrozumienie całego kodu na szczegółowym poziomie nie jest potrzebne. Aby nie musieć tego robić w systemach informatycznych stosuje się różnego rodzaju abstrakcje (system operacyjny, język programowania, protokoły). Konieczność rozumienia całego kodu oznacza źle zaprojektowany/napisany system.
2	Różne części systemu mogą być tworzone w różnych technologiach, co może utrudnić stosowanie automatów pomagających w procesie wytwórczym (np continuous integration).	Automaty pomagające w procesie wytwórczym mogą być niezależne dla każdego serwisu z osobna.
3	Automaty pomagające w procesie wytwórczym mogą być niezależne dla każdego serwisu z osobna. Co budzi niepokój, że nie ma testów dla całości.	Testy dla całości nie są wykluczone. Nie są przypisane do jakiegokolwiek serwisu, tylko do ich zestawu tworzącego platformę. Aplikacja może także zawierać testy integracyjne uruchamiające inne aplikacje i sprawdzające poprawność ich współpracy.
4	Możliwość tworzenie w różnych technologiach może spowodować używanie przestarzałych lub źle zaprojektowanych narzędzi.	Odpowiednie testy integracyjne (niezależne od implementacji) powinny wykryć złe funkcjonowanie.
5	Możliwość tworzenie w różnych technologiach może spowodować używanie używanie przestarzałych lub źle zaprojektowanych narzędzi, które wprowadzą dziury bezpieczeństwa.	Testy bezpieczeństwa mikroserwisu nie są istotnie trudniejsze od testów bezpieczeństwa aplikacji monolitycznej. Problemem jest jednak m. in. fakt, że należy

		<p>takich testów wykonać wiele, po jednym na każdy mikroserwis.</p>
6	<p>Niezależność mikroserwisów polegająca na łatwym przebiegu ich modyfikacji może być w niektórych sytuacjach postrzegana jako problem, na przykład wtedy, gdy potrzebna będzie zmiana większej części systemu.</p>	<p>Łatwość modyfikacji nie wpływa negatywnie na możliwość wykonywania większych zmian. Wygląda na to, że problem tkwi w śledzeniu przez jakie mikroserwisy używana jest część Y mikroserwisu X. Ten problem będzie rozwiązany przez dokładne specyfikacje API. W mikroserwisie Z nie korzystamy z mikroserwisu X, lecz z określonego API. Zmiana API będzie stanowić problem, jednak nie jest to przypadłość tylko mikroserwisów.</p>
7	<p>Trudniejsze zarządzanie transakcjami.</p>	<p>Problem tkwi w zapewnieniu atomowości operacji, które angażują wiele mikroserwisów. Kontrargumentem jest, że takich serwisów nie powinno być. Atomowe operacje powinny być realizowane wewnątrz jednego serwisu. Z tego powodu operacje atomowe powinny być małe.</p>
8	<p>W przypadku mikroserwisów API, dla których wprowadzanie zmian jest trudnym procesem, będzie znacznie więcej niż w przypadku monolitu.</p>	<p>Więcej standardowych, opisanych protokołów jest pozytywnym aspektem.</p>
9	<p>Wraz ze wzrostem każdej niezależnej aplikacji, wzrasta też zespół ją tworzący – może przyczynić się to do problemów w koordynacji. Łatwiej kierować jednym, sporym zespołem, tworzącym spójną całość i pracującym nad jednym kodem, niż nad wieloma zespołami pracującymi nad niezależnymi aplikacjami, które powinny tworzyć spójną całość.</p>	<p>Niezależne aplikacje nie muszą być tworzone przez jeden zespół. W momencie gdy osiągają stosunkową stabilność (wprowadzane zmiany nie zmieniają API w poważny sposób) zespoły mogą pracować oddzielnie.</p>
10	<p>Mikroserwisy są stosunkowo nowym stylem programowania, który wymaga bardziej skomplikowanych narzędzi niż monolityczny odpowiednik. Jest też mniej znany, dlatego na</p>	<p>Pisanie pojedynczego mikroserwisu to pisanie (w zamierzeniu małego - pod względem funkcji) programu,</p>

	i tak już małym rynku, znajduje się mniej specjalistów wdrożonych w system. Może więc generować większe koszty podczas tworzenia aplikacji, zwłaszcza na początku.	który ma API. Nie jest to nic nowego. Uruchamianie wielu takich programów w celu stworzenia funkcjonującego produktu może nastęczać trudności (z powodu swojej żmudności i konieczności odpowiedniej konfiguracji). Zadaniem zespołu tworzącego środowisko deweloperskie jest zapewnienie narzędzia, które w łatwy sposób uruchamia wiele mikroserwisów.
<b>L.p.</b>	<b>Zmapowane ryzyka, które należy zestawić z korzyściami</b>	
1	Trudniej jest dbać o bezpieczeństwo małych, rozproszonych systemów, niż o jeden spory system. Problem ten może też generować dodatkowe koszty.	
2	Testowanie integracyjne jest trudniejsze, niż w przypadku architektury monolitycznej.	
3	Większe zużycie pamięci operacyjnej, niż w przypadku rozwiązań monolitycznych.	
4	Komunikacja między serwisami będzie obciążać sieć. Obciążenie będzie większe, niż w przypadku rozwiązań monolitycznych, których komponenty mogą komunikować się za pomocą wspólnej pamięci.	

## Analiza argumentów

Argumenty za i przeciw wykorzystaniu wzorca mikroserwisów są porównywalnej siły.

Uwzględniając, że aplikacje uruchamiane na platformie będą wytwarzane przez niezależnych producentów zapewnienie swobody technologicznej jest ważnym aspektem. Bez takiej swobody zbiór potencjalnych producentów aplikacji zostałby ograniczony.

Oprócz EZD i innych, planowanych aplikacji, będzie potrzeba uruchamiania na platformie innych aplikacji. Postaci tych aplikacji nie sposób przewidzieć. Ograniczanie jej może ograniczyć możliwe do zrealizowania funkcje. Architektura mikroserwisów, przez zapewnienie swobody technologicznej, zapewnia małe ograniczenia.

Jednym z założeń Platformy jest zmiana, możliwość wymiany komponentów bez zmiany całego systemu oraz możliwość dodawania dodatkowych aplikacji. Architektura mikroserwisów dobrze wpisuje się w to założenie, ponieważ ułatwia niezależne zmiany poszczególnych serwisów.

Poważnym argumentem przeciw mikroserwisom jest trudność w zapewnieniu bezpieczeństwa. Eliminacja luk bezpieczeństwa wynikających ze stosowania różnych

technologii, z których niektóre mogą być wadliwe, wymaga nakładu pracy. Każdy serwis musi być sprawdzony, czy nie zawiera luk. Ten sam problem występowałby jednak (choć w mniejszej skali) także w sytuacji, gdyby moduły wielu producentów kooperowały w ramach jednego systemu - oba rozwiązania wymagają certyfikacji aplikacji pod kątem bezpieczeństwa.

Rekomenduje się wykorzystanie wzorca mikroserwisów w budowie platformy aplikacji.gov.pl. Aby wybrać to rozwiązanie konieczne jest przeprowadzenie dodatkowych konsultacji z zespołem odpowiedzialnym za architekturę chmurową i bezpieczeństwo docelowej platformy.

## Konsultacje z NASK

### Użycie pamięci operacyjnej

Przyjmując zamierzoną liczbę 1700 instytucji korzystających z systemu, na instytucję będzie przypadać około 3GB RAMu. Ta ilość jest za mała, aby uruchomić platformę w oparciu o mikroserwisy.

Prawdopodobne jest, że w początkowym okresie wdrożeń nie wszystkie 1700 instytucji będzie korzystać z platformy. Gdyby założyć mniejszą liczbę instytucji - 300 - na instytucję przypada około 17GB RAMu, co szacunkowo pozwala na uruchomienie czterech aplikacji.

Zdaniem naszym i NASK w kontekście powyższych obliczeń nie należy bardzo przejmować się złożonością pamięciową. Ważniejsze od złożoności jest zapewnienie możliwości skalowania pamięci wwyż (więcej RAMu w węźle obliczeniowym) jak i wszerz (więcej węzłów obliczeniowych).

Architektura mikrousług dobrze wpisuje się w wymaganie skalowalności pamięci, ponieważ wydajność mikrousługi może być zwiększona przez przydzielenie jej większej ilości zasobów (wzwyż) oraz przez uruchomienie większej liczby instancji danej mikrousługi (wszerz).