Rozdział 1

Wprowadzenie

1.1 Architektura oparta o mikrousługi

1.1.1 Architektura zorientowana na usługi

Koncept architektury mikrousług, mimo, że w ostatnich kilkunastu miesiącach stał się szeroko omawianym tematem, powszechnie uważanym za innowację, to rozszerza tylko podejście, które istnieje od dawna jako SOA (ang. *Service-Oriented Architecture*). SOA to architektura łącząca wiele komponentów, których komunikacja odbywa się nie w granicach procesu, ale również przez sieć komputerową. Ściślej mówiąc, komunikacja powinna niejako ignorować fakt fizycznej lokalizacji procesu wywołującego jak i wywoływanego. System w ten sam sposób powinien zachować się zarówno uruchomiony na jednej maszynie jak i na wielu maszynach mieszczących się na różnych kontynentach.

Taka granulacja umożliwia lepsze wykorzystanie zasobów, poprzez dedykowanie ich konkretnym komponentom w zależności od nakładu pracy, który jest wykonywany. Stosuje się podejście tworzenia wielu drobnych (*ang. fine-grained*) aplikacji przeznaczonych do jednego zadania.[**?**] Pozwala to na skalowanie tylko obciążonej części przetwarzania, aby udrożnić wąskie gardło systemu.

1.1.2 Domain Driven Design jako źródło mikrousług

Procesem rozwoju oprogramowania rządzi szereg podstawowych zasad, które umożliwiają skuteczne pisanie, a przede wszystkich utrzymywanie istniejącego kodu. Jedną z najważniejszch w ocenie autora reguł jest zasada pojedynczej odpowiedzialności, stanowiąca o tym, że każda funkcjonalność powinna być realizowana przez jedną wydzieloną część kodu i tylko przez nią. Na poziomie programowania zorientowanego obiektowo jest to klasa, na poziomie projektu jest to pakiet, a na poziomie architektury może być to komponent. Koncepcja projektowania aplikacji w oparciu o tę zasadę została nazwana zasadą projektowania sterowanego dziedziną (ang. *Domain Driven Design*)[**?**] i ostatnio bardzo często spotykanym paradygmatem prezentowania rzeczywistego świata w kodzie źródłowym.

Według tego podejścia, cały proces wytwarzania oprogramowania, począwszy od zbieranie wymagań, przez analizę, projektowanie, implementację, aż po testowanie i wdrożenie powinno być oparte o wiedzę na temat wycinka rzeczywistości, który mapowany jest na oprogramowanie w celu ułatwienia lub przyspieszenia rozwiązywania problemów z nim związanych.

1.1.3 Zasada autonomiczności jako źródło problemów

Tworzenie mikrousług zgodnie z zaleceniami twórców tego konceptu wymaga również zmian w strukturach organizacji, która wytwarza system. Zmiana taka powinna polegać na zorganizowaniu całych zespołów wokół każdej usługi. Oczywiście każdy zespół może realizować prace związane z wieloma usługami, ale ważne jest aby jedna usługa była rozwijana i ustrzymywana przez jeden zespół. Podejście takie wymaga stworzenia kilkuosobowych zespołów, w których znajdują się

graficy, testerzy, projektanci i programiści. Zespół ten sam decyduje, kiedy wdroży nową wersję swojej usługi.

Problem pojawia się w momencie, kiedy niezależny zespół wprowadza zmiany, które wymagają zmian również po stronie konsumentów. Są to tzw. *zmiany przełomowe[?].* Podczas projektowania interfejsów należy statrać się jak najrzadziej wykonywać takie zmiany. Wymaga to przede wszytkim bardzo dokładnej analizy dziedziny problemu, jaki rozwiązuje dana usługa, aby uwzględnić kierunki, w których może się ona rozwijać. Unikanie zmian nie zawsze będzie możliwe, ale nie każda z nich jest zmianą przełomową. Np. dodanie pola do zwracanego obiektu nie powinno zmienić zachowania konsumanta. W tym miejscu należy też wspomnieć o tym, aby konsumentów implementować w sposób „odporny" na zmiany, które nie są przełomowe.

Niezależnie rozwijane komponenty mimo niezaprzeczalnych korzyści jakie niosą, mają też wady i z pewnością jest to dbanie o kompatybilność z oprogramowaniem z nich korzystającym.

1.2 Komunikacja między usługami

NOTE: Aby ta komunikacja była uporządkowana, warto zastosować konwencję, która definiowała będzie sposób odwoływania się do zasobów

1.3 Język RAML

1.4 Testowanie jednostkowe

[Samo przepisywanie kodu na mikrousługi jest wystarczająco trudny więc warto zaoszczędzić czas na testeowaniu] [S. G. Saez, V. Andrikopoulos, F. Wessling, and C. C. Marquezan, "Cloud Adaptation and Application (Re-)Distribution: Bridging the Two Perspectives," in 2014 IEEE 18th International Enterprise Distributed Object Computing Conference Workshops and Demonstrations. IEEE, Sep. 2014, pp. 163–172. [Online]. Available: http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=6975357]