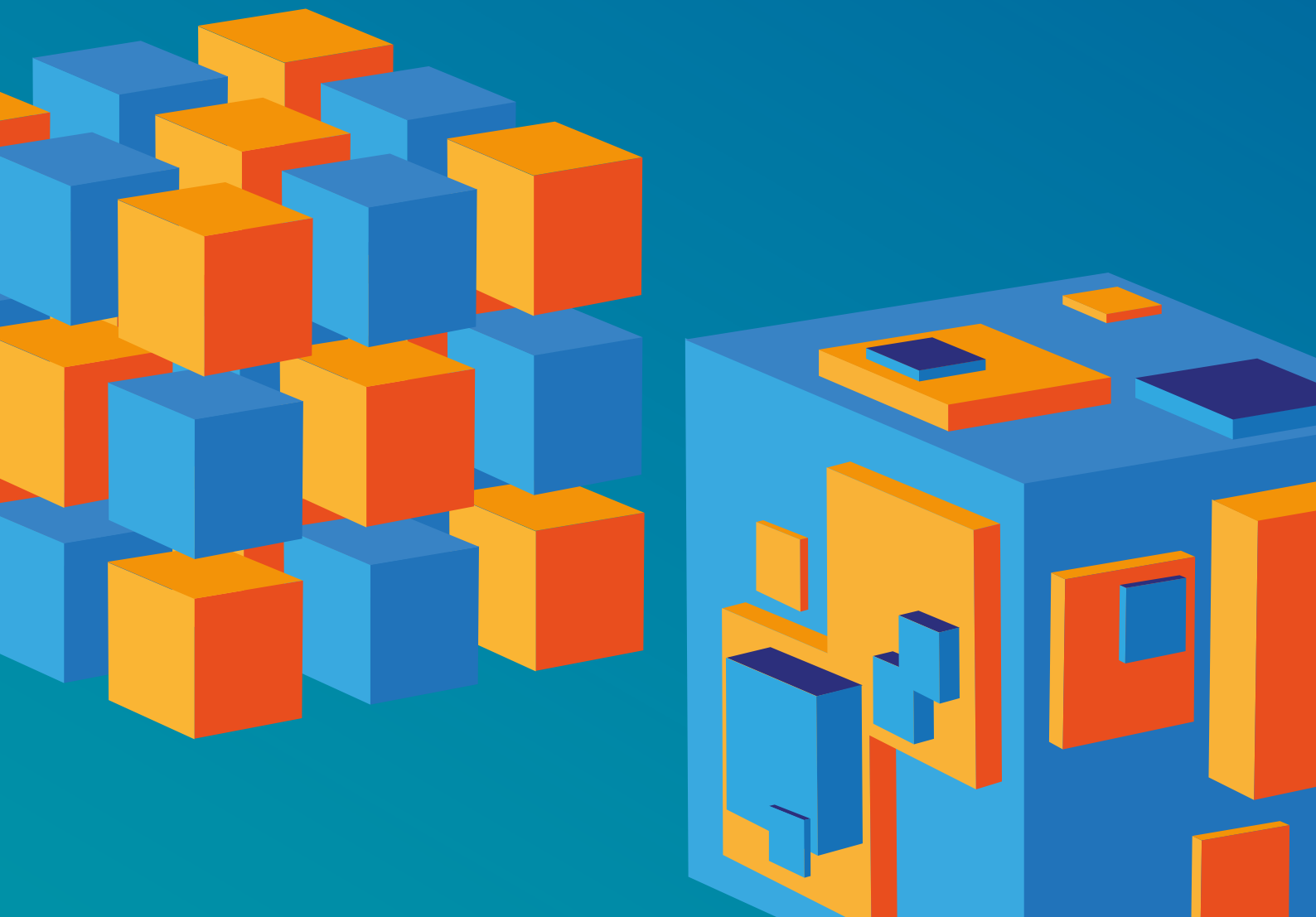
RST SOFTWARE MASTERS

Od monolitu do mikroserwisów,

czyli jak wyjść ze struktury hamującej rozwój



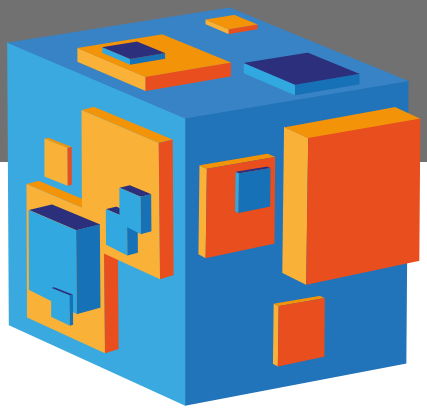
DEFINICJA MIKROSERWISÓW

Próbując zdefiniować mikroserwis można przyjąć bardzo konkretne kryteria. Przykładowo: „mikroserwis to komponent zawierający nie więcej niż 100 wierszy kodu" albo „zawierający nie więcej niż jedną metodę". Tylko czy usługa składająca się ze 101 wierszy kodu to już nie mikroserwis? A może górna granica wynosi 110 wierszy kodu?

Skoro próba wyznaczenia liczbowej granicy nie prowadzi do jednoznacznych wniosków, poszukajmy przykładu odwołującego się do wyobraźni. Co jest tak duże, że na pewno nie zasługuje na miano mikroserwisu? Weźmy jako przykład panel CRM (Customer Relationship Management). Taki panel realizuje zazwyczaj mnóstwo funkcji.

PANEL CRM JAKO MONOLIT

Monolit stanowi przeciwieństwo mikroserwisu.
Jest to duży komponent realizujący wiele funkcji z różnych obszarów biznesu.



Nie dość, że funkcji jest sporo, to należą one do różnych obszarów biznesu. Warto tutaj odwołać się do znanego z DDD (domain-driven design) pojęcia „bounded context" oznaczającego jeden z obszarów działalności. Przykładowo, wysyłanie powiadomień e-mail należy do innego kontekstu niż ochrona przed nieuczciwymi klientami i ich blokowanie. Raporty wspierają sprzedaż, ale nie mają nic wspólnego ze zmianą haseł do kont pracowników. Panel CRM nie jest więc mikroserwisem, bo realizuje funkcje należące do różnych obszarów biznesu.

Poza rozmiarem i liczbą funkcji, w ustaleniu czym jest mikroserwis może pomóc czas potrzebny na jego przygotowanie.

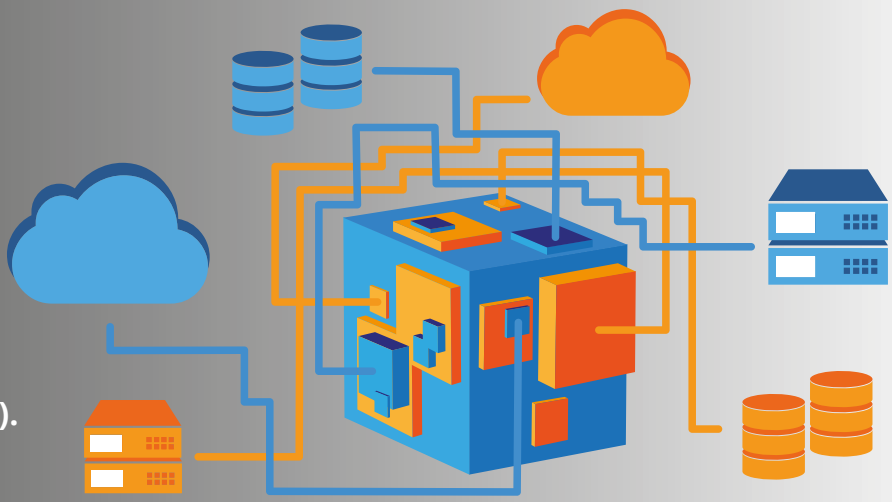
W najprostszym przypadku od pomysłu do udostępnienia mikroserwisu mogą upłynąć zaledwie pojedyncze minuty. Z drugiej strony, jeśli powstanie pierwszej działającej wersji usługi zajmuje więcej niż kilka tygodni to najwyraźniej nie zasługuje ona na miano mikroserwisu.

Mikroserwis jest niewielką rozmiarem usługą, realizującą funkcje z pojedynczego obszaru biznesu, której przygotowanie trwa nie więcej niż kilka tygodni.

MONOLIT KONTRA MIKROSERWIS

W architekturze mikroserwisowej wprowadzenie nowej funkcji odbywa się przez zmianę jednego z istniejących mikroserwisów lub dodanie nowego. Wdrożenie zmian sprowadza się często do wdrożenia jednego mikroserwisu. Z kolei wprowadzenie zmiany do monolitu jest procesem bardziej skomplikowanym, bo aby udostępnić nawet niewielką nową funkcję trzeba wdrożyć cały monolit.

Kod monolitu znajduje się zazwyczaj w jednym repozytorium, a różne funkcje realizowane przez monolit korzystają ze wspólnej infrastruktury (baz danych, serwerów HTTP itd.).



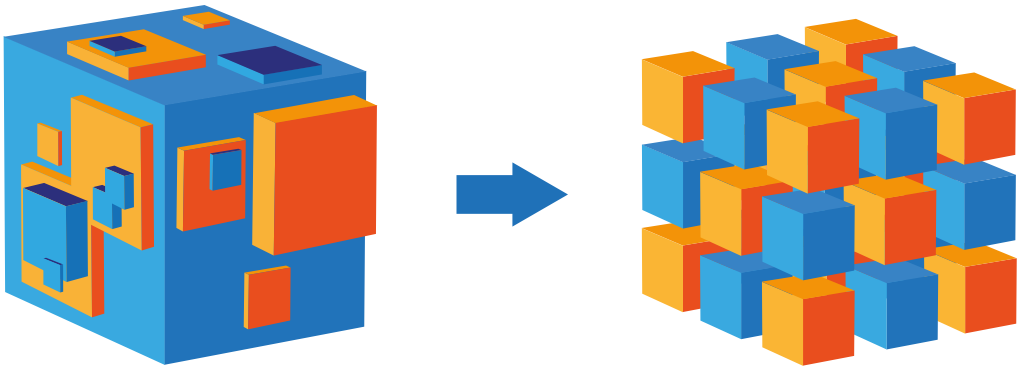
Panel CRM może być znakomitym przykładem monolitu, o ile wszystkie jego funkcje są realizowane przez jeden komponent. Aby zrealizować panel CRM w architekturze mikroserwisowej, musimy rozbić jeden duży komponent na wiele mniejszych, z których każdy jest zgodny z definicją mikroserwisu. Z jednej strony mamy CRM w postaci monolitu, z drugiej – rozbity na mikroserwisy.

A może jest też droga pośrednia dla organizacji chcących migrować z jednej architektury do drugiej? Praktyka pokazuje, że tak. Droga ta polega na stopniowej migracji i ograniczaniu roli monolitu poprzez wydzielanie z niego kolejnych mikroserwisów. Taką właśnie drogę przyjęliśmy planując rozwój systemu Trans.eu.

Jedną z głównych różnic między mikroserwisami a monolitem jest sposób wprowadzania zmian.

PRZEJŚCIE OD MONOLITU DO MIKROSERWISÓW OCZAMI PRAKTYKA

System Trans.eu przez kilkanaście lat rozwijaliśmy w formie monolitu. Przez ten czas repozytorium kodu znacznie się rozrosło i coraz trudniej było sprawnie z nim pracować. Patrząc na różne fragmenty kodu można było obserwować, jak z biegiem czasu zmieniały się dobre praktyki inżynierii oprogramowania. Nowsze fragmenty były dobrze pokryte testami automatycznymi i wprowadzanie w nich zmian było stosunkowo proste. Z kolei starsze fragmenty nie były pokryte takimi testami, przez co wymagały bardziej ostrożnego podejścia i wykonywania ręcznych testów regresyjnych. Te różnice przekładały się na trudne do oszacowania koszty wprowadzania zmian.

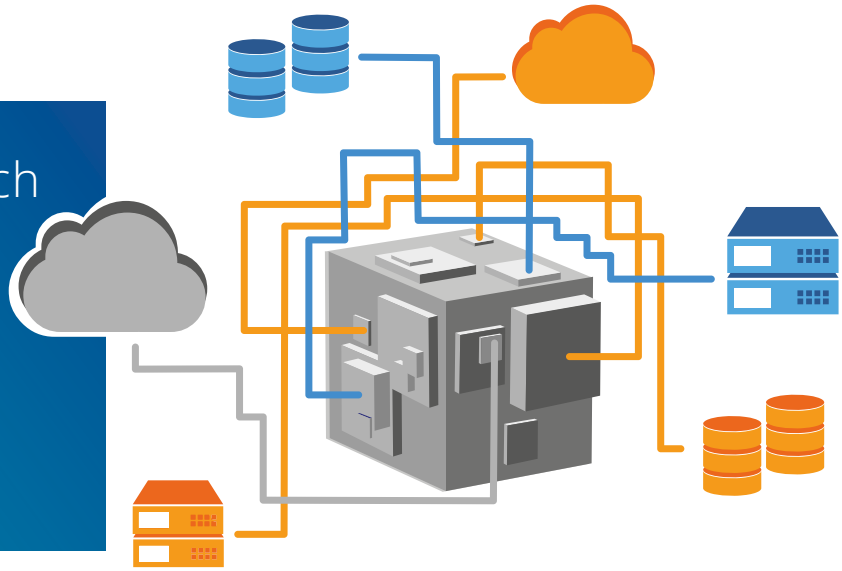


Jako że zmiana w monolicie mogła potencjalnie wpłynąć na dowolną jego część, wdrażanie nowych wersji było utrudnione. Za każdym razem trzeba było upewnić się, że całość działa poprawnie, co było skomplikowanym procesem wymagającym zaangażowania przedstawicieli wielu zespołów deweloperskich. Od rozpoczęcia realizacji wymagań do wdrożenia upływało co najmniej kilka tygodni. Biznes miał ograniczoną możliwość eksperymentowania z nowymi funkcjami i reagowania na uwagi od użytkowników. Wspólnie z klientem doszliśmy do wniosku, że jako organizacja potrzebujemy stać się bardziej elastyczni.

Skupienie wszystkich funkcji systemu w jednym miejscu powodowało też, że pojedynczy błąd potrafił dotknąć kilku różnych obszarów systemu. Bardzo wyraźnie widzieliśmy potrzebę odizolowania od siebie poszczególnych funkcji realizowanych przez monolit.

W pracę nad jednym repozytorium kodu było zaangażowanych kilkanaście zespołów rozmieszczonych w dwóch oddziałach firmy. Działając w ten sposób, przymykaliśmy oko na prawo Conwaya.

W skrajnych przypadkach awaria jednej z funkcji monolitu powodowała, że cały monolit stawał się niedostępny dla użytkowników.



Mówi ono, że organizacja pracując nad systemem (w szczególności – informatycznym) odwzorowuje w nim swoją strukturę. Jak to się objawiało na naszym przykładzie? Stosowaliśmy różne technologie, ale monolit był tworzony tylko w jednej z nich. To skutkowało narzutem na poznanie wspólnej technologii nawet w tych zespołach, które na co dzień nie miały z nią wiele wspólnego. W dodatku jakość komunikacji (np. niewielka fizyczna odległość między danymi zespołami) znajdowała swoje odzwierciedlenie w strukturze naszego monolitu.

Skoro różnice pomiędzy poszczególnymi częściami organizacji były tak łatwo zauważalne, to czy opłacało się je ignorować?

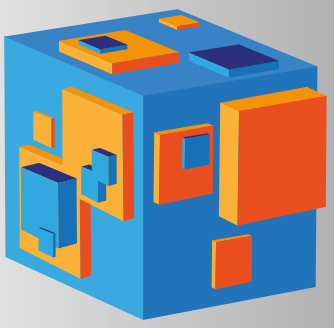
Mikroserwisy pozwalają przekuć taką różnorodność w korzyści.

Koszt rozwoju systemu jest niższy, kiedy zespoły mogą skupić się na najbliższej sobie technologii i tym obszarze biznesu, który rozumieją najlepiej.

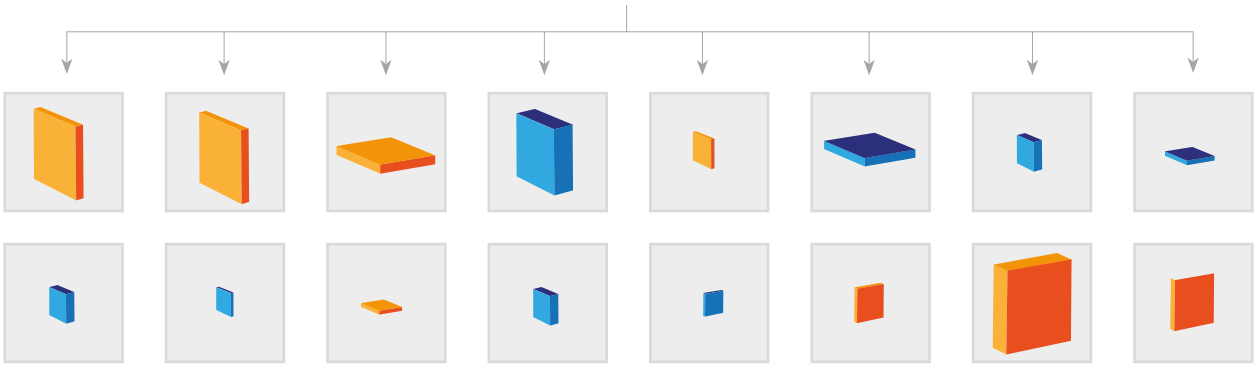
Zamiast podejmować decyzję o migracji w ciemno i przez kilka lat przechodzić obojętnie wobec szans na rozwój biznesu, zaproponowaliśmy stopniowe przejście.

OD CZEGO ZACZĘLIŚMY?

Gdybyśmy chcieli w jednym kroku przekształcić cały monolit na mikroserwisy, musielibyśmy poświęcić na to kilka lat. Rozwój produktu trzeba by ograniczyć do minimum, inaczej każdą zmianę musielibyśmy nanosić i na monolit i na mikroserwisy, jeszcze bardziej wydłużając migrację. Takie podejście nie wchodziło w grę.



Aby przekształcić monolit na mikroserwisy, należy rozbić jeden duży komponent na wiele mniejszych, z których każdy jest zgodny z definicją mikroserwisu.



Powstała pula serwerów dedykowanych do uruchamiania mikroserwisów. Zapewniała ona proste środowisko, dzięki któremu mogliśmy szybko zacząć migrację. Żeby bardziej skupić wysiłek i możliwie najwcześniej wprowadzać w życie wnioski z eksperymentu, skupiliśmy się na kilku mikroserwisach. Wśród nich znajdowały się takie do uwierzytelniania użytkowników, wysyłki SMS-ów i aktualizacji pozycji ciężarówek na mapie.

Czas od powstania koncepcji biznesowej do wdrożenia działającego oprogramowania znacznie się skrócił. Monolit wdrażaliśmy zaledwie raz na kilka tygodni, podczas gdy mikroserwisy były wdrażane nawet kilka razy tego samego dnia. Skutki ewentualnych błędów były ograniczone do wdrażanego mikroserwisu, a zespoły zaczęły poszukiwać narzędzi do tworzenia mikroserwisów w technologiach, które znały najlepiej.

Wiedzieliśmy, że mikroserwisy tworzone w ramach eksperymentu nie były doskonałe, a środowisko ich działania stanowiło raczej punkt wyjścia niż gotową platformę. Udało się jednak potwierdzić słuszność kierunku i zebrać listę obszarów do dalszego rozwoju.

Bazując na wypracowanych wcześniej dobrych praktykach i wynikach eksperymentu, ustaliliśmy co jest potrzebne do sprawnej pracy w architekturze mikroserwisowej.

WYMAGANIA DLA MIKROSERWISÓW

Na liście wymagań dla mikroserwisów znajdują się:

Szybkie tworzenie środowiska dla nowych mikroserwisów

Powinno się ono odbywać w sposób zautomatyzowany, ewentualnie z niewielkim udziałem człowieka. Trudna do zaakceptowania jest sytuacja, w której trzeba z wyprzedzeniem planować środowisko dla nowego mikroserwisu, składać zamówienie (np. na fizyczny sprzęt) i czekać dłużej niż trwałby sam development.

Automatyczne wdrażanie mikroserwisów w jednym kroku

Wprowadzanie zmian w mikroserwisach jest prostsze niż w monolicie. Warto te zmiany wdrażać możliwie najczęściej, a żeby to było możliwe, potrzeba automatycznego procesu wdrożeniowego, który można wykonać w jednym kroku. Pomóc może np. zastosowanie standardowych mechanizmów paczkowania udostępnianych przez systemy operacyjne.

Dokumentacja architektury powiązań

Powinna zawierać zarys architektury mikroserwisu z uwzględnieniem zależności od innych mikroserwisów. Jeżeli utrzymaniem zajmuje się inny zespół niż ten który tworzył mikroserwis, przydatne będą konkretne wskazówki (np. zasady postępowania w sytuacjach awaryjnych).

Wygodnie jest jeśli dokumentacja jest dostępna w repozytorium kodu mikroserwisu i można ją otworzyć bez użycia specjalistycznego oprogramowania.

Identyfikacja zależności

Oprócz opisania zależności w dokumentacji mikroserwisu, można przechowywać je w formie nadającej się do automatycznego przetwarzania. Da się dzięki temu stworzyć narzędzie wspomagające uzyskanie odpowiedzi na pytania takie jak: czego dany mikroserwis potrzebuje do działania? Jakie mikroserwisy potrzebują realizowanych przez niego funkcji? Kto powinien zostać powiadomiony o nowej wersji API? Które mikroserwisy są kluczowe dla sprawnego działania systemu i potrzebują więcej zasobów niż pozostałe?

Niezależność mikroserwisów od siebie nawzajem

Jeśli któryś z mikroserwisów nie będzie dostępny (np. na skutek awarii), to ucierpieć powinny jedynie realizowane przez niego funkcje. Niedopuszczalna jest sytuacja, w której niedostępność będzie się propagować na wiele mikroserwisów.

Niezależność można osiągnąć rezygnując z synchronicznych zależności między mikroserwisami na rzecz wymiany wiadomości np. przez protokół AMQP. Można też budować read modele, tj. lokalne kopie danych potrzebnych mikroserwisowi, a pochodzące od innych mikroserwisów.

Istotna jest też niezależność infrastruktury. Każdy mikroserwis powinien korzystać z własnych baz danych czy też np. silników wyszukiwania. Dzięki temu można izolować skutki ewentualnych problemów z infrastrukturą.

Niezależność od konkretnych instancji mikroserwisu

W celu zapewnienia wysokiej dostępności, każdy mikroserwis powinien być uruchomiony w przynajmniej dwóch instancjach. Szczegóły techniczne takie jak adresy IP tych instancji nie powinny być istotne dla pozostałych mikroserwisów. Aby to osiągnąć, można wdrożyć narzędzia do service discovery. Wyszukiwanie mikroserwisu odbywa się wtedy na podobnej zasadzie jak wyszukiwanie domen w systemie DNS.

Standardowe sposoby komunikacji

Najczęściej spotykanym sposobem komunikacji między mikroserwisami jest REST, czyli protokół HTTP wzbogacony o zestaw konwencji i dobrych praktyk. W ramach zwiększania niezależności pojawiają się często protokoły do wymiany wiadomości, takie jak AMQP. Szczegóły komunikacji z mikroserwisem powinny być opisane w jego dokumentacji.

Niezależnie od przyjętych sposobów, ważne by ich stosowanie było ujednolicone w całym systemie. Sposób komunikacji nie powinien zależeć od tego który zespół tworzył mikroserwis ani tego jaka technologia została użyta.

Możliwość monitorowania

Obserwowalny mikroserwis dostarcza informacji na temat swojego stanu. Informacje takie mogą mieć postać metryk wizualizowanych na wykresach, a także wpisów w logach systemowych. Obserwowalność zwiększa szansę zapobiegania awariom i ułatwia znajdowanie ich przyczyn.

Prowadzenie dziennika zmian

Lista zmian wprowadzanych do mikroserwisu. Dziennik zmian jest szczególnie przydatny dla użytkowników mikroserwisu, którymi są zazwyczaj inne zespoły deweloperskie. Umożliwia śledzenie nowych funkcji i poprawek błędów.

Wskazanie właściciela

Właściciel (zespół deweloperski) może być wskazany w dokumentacji mikroserwisu lub w innym, dedykowanym i dostępnym dla wszystkich miejscu. Dzięki możliwości łatwego ustalenia właściciela mikroserwisu wiadomo do kogo się zwrócić z prośbą o dodanie funkcji bądź naprawienie błędu.

Jeżeli za utrzymanie mikroserwisu jest odpowiedzialny inny zespół niż jego właściciel (np. gdy utrzymaniem zajmuje się zespół administratorów), to dzięki wskazaniu właściciela skrócony zostaje czas potrzebny na uzyskanie potrzebnych informacji w razie awarii.

Szkielety do tworzenia mikroserwisów

Szkielety powinny zawierać gotowy zestaw narzędzi niezbędnych do spełnienia wymagań stawianych przed mikroserwisami. Celem istnienia szkieletu jest oszczędzanie czasu potrzebnego na dostosowanie się do standardów, a także zwiększenie jednolitości w systemie.

Szkielety nie są niezbędne, ale stanowią inwestycję która szybko się zwraca.

Powyższe wymagania służą temu,
by praca w architekturze mikroserwisowej
przebiegała zgodnie z oczekiwaniami:
sprawnie, przewidywalnie i z możliwością
częstego wdrażania zmian.

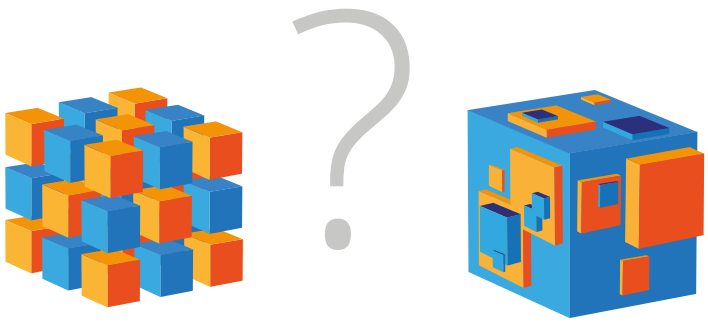
KOMU POLECAMY MIKROSERWISY?

Jakie okoliczności powodują, **że warto** przejść z monolitu na architekturę mikroserwisową?

- Różnorodność w obrębie organizacji: wiele firm lub zespołów pracujących nad jednym systemem, wiele oddziałów, różne technologie.
- Rozbudowany zakres funkcji realizowanych przez system.
- Potrzeba łatwego i częstego konfrontowania pomysłów z odbiorcami.
- Praca nad systemem który był lub będzie rozwijany przez wiele lat i ma przetrwać zmiany w technologii i trendach inżynieryjnych.
- Różnorodne wymagania dot. skalowania systemu, tj. sytuacja w której pewne funkcje są używane dużo częściej i przez większą liczbę użytkowników niż pozostałe.

Jakie okoliczności powodują, **że nie warto** stosować architektury mikroserwisowej?

- System który jest tylko utrzymywany, a nie rozwijany.
- Zupełnie nowe przedsięwzięcie (startup), co do którego nie ma jeszcze pewności czy warto inwestować w dobre praktyki.
- Rozwijanie systemu przez jeden – dwa zespoły, które mogą się komunikować bezpośrednio.
- System z bardzo ścisłymi wymaganiami dot. czasu odpowiedzi, w którym dodatkowy narzut sieciowy jest niedopuszczalny.



Szkoda czasu na pracę w architekturze hamującej rozwój.

KROPKA NAD I

Architektura mikroserwisowa spełniła pokładane w niej nadzieje na zniesienie barier w rozwoju systemu Trans.eu. Sprawdzi się ona w każdej organizacji, która potrzebuje technicznej i biznesowej dojrzałości. Potrzeba częstego konfrontowania się z rynkiem, eksperymentowania i skalowania tylko wzmacnia możliwe do osiągnięcia korzyści.

Aurtor opracowania:

Łukasz Wróbel

IT Architect

Absolwent Informatyki na Uniwersytecie Wrocławskim. Doświadczony developer, team leader, architekt i menadżer. Pochłonięty projektowaniem wydajnych i skalowalnych aplikacji internetowych i tworzeniem kodu wysokiej jakości. Swoją wiedzą dzieli się podczas konferencji branżowych i warsztatów.

Nie wiesz, jak zacząć zmieniać swój system?
Jeśli masz pytania, chętnie odpowiem.

Justyna Gieracka
Business Development Representative

+48 532 626 248
justyna.gieracka@rst.com.pl

RST SOFTWARE MASTERS

RST SOFTWARE MASTERS
ul. Racławicka 2-4, 53-146 Wrocław
rst.com.pl