Paweł Rajba
pawel@ii.uni.wroc.pl
http://www.itcourses.eu/

MicroServices

Agenda

- Wprowadzenie
- Design Principles
- Technologie
- Kto tego używa?

Wprowadzenie

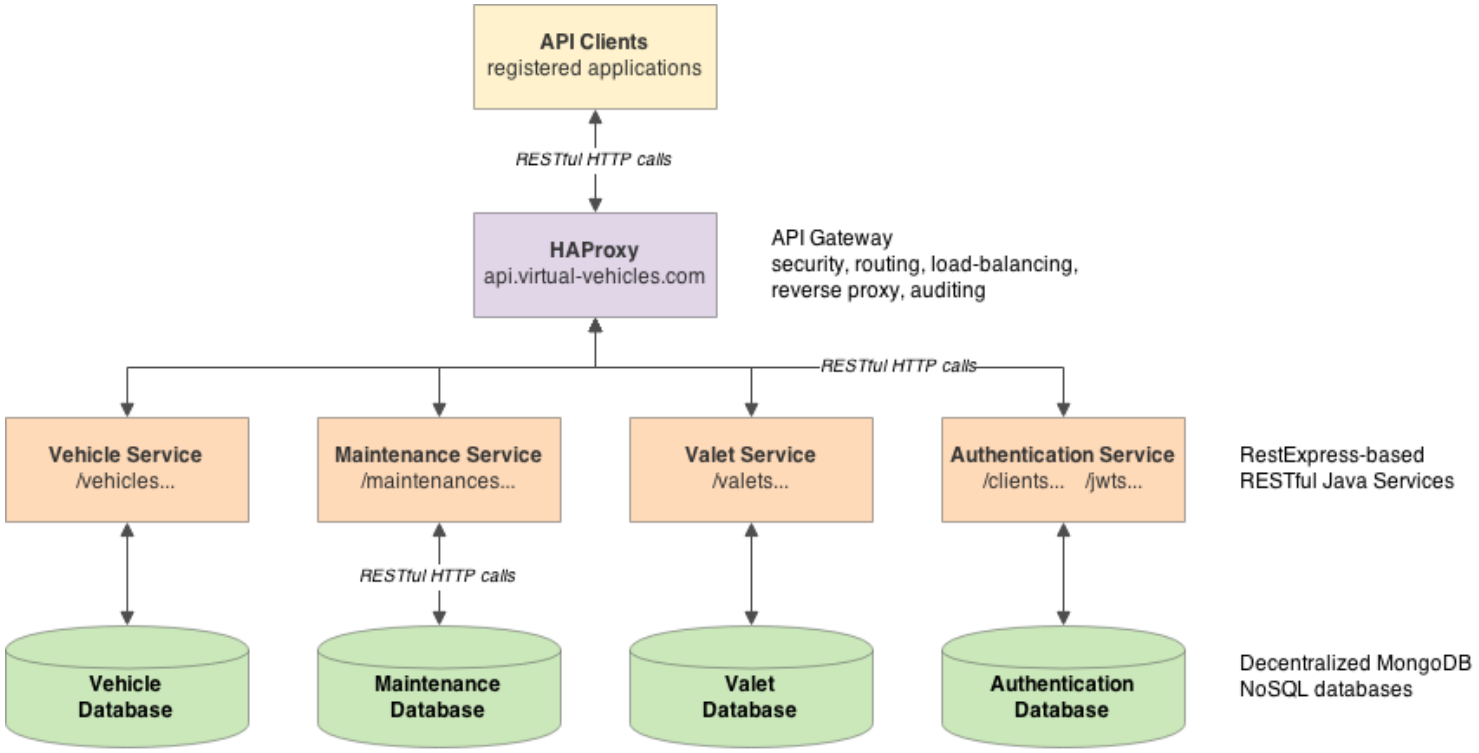
- Service Oriented Architecture
  - Funkcjonalności udostępnione jako usługi
  - Określony kontrakt wymiany danych
  - Daje możliwość skorzystania z funkcjonalności wielu różnym klientom (web, mobile, desktop)
  - Dobrze napisane umożliwia łatwe skalowanie
  - Cykl życia usługi nie wymusza zmian u klientów (dopóki zachowany jest kontrakt)

Wprowadzenie

- Microservices
  - Koncept SOA, ale w bardziej granularnej postaci
  - „Małe” usługi z pojedynczymi odpowiedzialnościami
  - Aplikacja zwykle bazuje na wielu usługach
  - Dostęp do usług nie powiązany z żadną technologią
  - Lekka komunikacja pomiędzy komponentami
  - Każda usługa ma własne repozytorium danych
  - Każda usługa ma własny cykl życia i publikacji
  - Transakcje najczęściej są rozproszone
    - … lub oparte o „eventual consistency”

Wprowadzenie

- Microservices, przykładowy diagram



https://programmaticponderings.files.wordpress.com/2015/05/virtual-vehicles-architecture-4.png

Wprowadzenie

- Wady monolitycznej architektury
  - Czas publikacji dużego systemu może być długi
  - Publikacja obarczona większym ryzykiem błędów
  - Małe, szybkie zmiany muszą czekać na „release”
  - Sztywny zbiór technologii (technology stack)
    - Brak możliwości łatwej adaptacji nowych technologii
  - Często silne powiązania pomiędzy komponentami
    - Jeśli kod jest źle napisany, a najczęściej tak jest
  - Skalować musimy cały system, a nie pojedyncze składowe, które potrzebują skalowania

Wprowadzenie

- Dlaczego MicroServices?
  - Szybsza reakcja na zmiany w biznesie, częstsze publikacje
  - Uniezależnienie często zmienianych komponentów
  - Łatwiejsza skalowalność
  - Większa odporność i bezpieczeństwo systemu
    - Awaria w jednej usłudze nie musi oznaczać awarii całego systemu
  - Łatwiej odwzorować strukturę funkcjonalną organizacji (DDD)
  - Sporo wsparcia w technologii
    - Lekkie protokoły
    - Usługi w chmurze, wirtualizacja
    - Wsparcie dla komunikacji asynchronicznej

Wprowadzenie

Microservices provide benefits…

- Strong Module Boundaries: Microservices reinforce modular structure, which is particularly important for larger teams.

- Independent Deployment: Simple services are easier to deploy, and since they are autonomous, are less likely to cause system failures when they go wrong.

- Technology Diversity: With microservices you can mix multiple languages, development frameworks and data-storage technologies.

…but come with costs

- Distribution: Distributed systems are harder to program, since remote calls are slow and are always at risk of failure.

- Eventual Consistency: Maintaining strong consistency is extremely difficult for a distributed system, which means everyone has to manage eventual consistency.

- Operational Complexity: You need a mature operations team to manage lots of services, which are being redeployed regularly.

Design Principles

- Wysoka spójność
  - SOLID: Single Responsibility Principle
    - tylko jeden powód na zmianę usługi
  - Ten „powód” może wynikać z obszaru lub funkcji biznesowej
    - Ważne, żeby go jasno ustalić
  - Jeśli trzeba:
    - tworzymy nowe usługi
    - przebudowujemy istniejące

Design Principles

- Autonomiczność
  - Luźne powiązania oparte na kontraktach i interfejsach
  - Komunikacja
    - Synchroniczna
    - Asynchroniczna (np. przez publikację i obsługę zdarzeń)
  - Niezależność technologiczna
    - Unikanie publikacji bibliotek klienckich. Wyjątki?
  - Unikamy współdzielenia baz danych i bibliotek
  - Usługa ma własny cykl życia i zmiany w innych komponentach powinny być niezależne
  - Wsteczna kompatybilność (Open-Closed principle)
  - Strategia wersjonowania
    - Utrzymywanie dwóch wersji przez pewien czas
      - Warto przemyśleć odpowiednie endpointy (np. /V2/invoice/)
    - Semantic versioning: Major.Minor.Patch (np. 1.7.12)

Design Principles

- Business Domain Centric
  - Granice funkcjonalności powinny być wyznaczone przez funkcję biznesową
    - Związek z Bounded Context z DDD
    - Wsparcie dla Domain lub Sub Domain
    - Granice powinny się dostosowywać wraz z rozwojem domain lub subdomain
  - Jasny interfejs i kontrakt „na zewnątrz”
  - Aspekt techniczny
    - Niepożądany, ale czasami pragramatyczny
      - Problem z wydajnością
      - Osobna usługa do archiwizacji danych

Design Principles

- Odporność na awarie
  - Przygotować system na spodziewane problemy
    - Np. wynikające z interakcji z usługami zależnymi
    - Przykładowe strategie to
      - Wyłączyć ten rodzaj funkcjonalności np. pokazywanie promocji
      - Udostępnić podstawową lub ograniczoną funkcjonalność (np. ceny domyślne bez rabatów)
  - Fail fast
    - Jeśli coś idzie nie tak, szybki feedback do użytkownika
    - Nie przeciągać transakcji
    - Sterowanie przez timeouty (globalne, per transakcja, itp.)
  - Problemy mogą być na wielu poziomach: wyjątki, opóźnienia, niedostępność, problemy z siecią
    - Monitorować timeouty do późniejszej analizy

Design Principles

- Monitoring i logging
  - Status systemu (health check)
  - Zcentralizowany monitoring
    - Monitorowanie na różnych poziomach
      - Infrastruktury, transakcji biznesowych
    - Monitorowanie różnych aspektów
      - Timeouty, czasy odpowiedzi, dostępy do danych, procesy biznesowe (np. czas od zatwierdzenia koszyka do złożenia płatności)
    - Określenie SLA i generowanie powiadomień w przypadku jej niedotrzymania
    - Narzędzia do wizualizacji danych
    - Rozwiązywanie problemów i planowanie skalowania
  - Zcentralizowany logging
    - Rejestracja istotnych zdarzeń, aby sprawnie rozwiązywać problemy
    - Rejestracja odpowiednich danych: czasu, hosta, komunikatu, itd.
    - Rejestracja CorrelationID do śledzenia rozproszonych transakcji

Design Principles

- Automatyzacje
  - Continues Integration
    - Wykonanie unit i integration tests
  - Continues Deployment, DevOps
    - Daje bardzo szybką publikację zmian
    - Współpracuje zwykle z CI
    - Użytkownik bardziej zadowolony

Technologie

- Komunikacja
  - Synchroniczna
  - Asynchroniczna
    - Zdarzenia i kolejki, platforma integracyjna
  - Oparta o RPC, HTTP, REST
- Hosting
  - Wirtualizacja
  - Containers
    - Wirtualizacja w bardziej granularnym wymiarze np. Docker
  - Własne maszyny lub IaaS

Technologie

- Rejestracja i katalog
  - Pożądanym jest posiadanie repozytorium dostępnych usług
  - Usługa może rejestrować się sama, rejestracja może być też zapewniona przez infrastrukturę
  - Infrastruktura może monitorować usługi i wyrejestrowywać te, które nie odpowiadają
- Monitoring
  - Na poziomie sieci: Nagios, PRTG
  - Na poziomie aplikacji: New Relic, BMC APM
- Logging
  - Rozwiązania: Splunk, Kibana, ważne automatyczne analizy
  - Śledzenie transakcji poprzez CorrelationID
  - Kwestia formatu logów
  - Dla MicroServices: koniecznie zcentralizowany

Technologie

- Skalowanie
  - Więcej instancji usługi vs. więcej zasobów dla usługi
  - Automatyczne vs. na żądanie
  - Load balancers
  - Kiedy: problemy z wydajnością (bieżące lub spodziewane)
- Caching
  - Są dużym wsparciem dla poprawy wydajności
  - Implementowane na różnych poziomach (proxy, client)

Technologie

- API Gateway
  - Główny punkt wejścia dla infrastruktury MicroS.
  - Może implementować
    - Load balancing, Caching, Security
  - Pozwala na
    - Jeden punkt dostępowy do wszystkich MicroServices
    - Bardziej dynamiczne zarządzenia infrastrukturą MS
    - Lepszy routing, np. klientów z Europy kierować do odpowiednich serwerów

Kto tego używa?

- Comcast Cable
- Uber
- Netflix
- Amazon
- Ebay
- Sound Cloud
- Karma
- Groupon
- Hailo
- Gilt
- Zalando
- Capital One Why Capital One is at Re:Invent and Keynote
- Lending Club
- AutoScout24

How we build microservices at Karma

"Microservices" and "Microservice Architecture" are hot buzz words in the development community right now, but concrete examples of microservices in production are still scarce. I thought it might help to give a brief overview of how we've utilized microservices for our backend API at Karma over the past couple of years. It's not exactly a "how-to," more like a "why-to," or a "wherefore-to," but hopefully it will give you some insight as to whether microservices are appropriate for your application, and how you can go about using them.

the UK Government Digital Service
Real Estate
Property & Homes For Sale
Forward
Twitter
PayPal
Bluemix
The Guardian

Literatura

- http://martinfowler.com/articles/microservices.html
- http://martinfowler.com/microservices/
- https://smartbear.com/learn/api-design/what-are-microservices/
- https://en.wikipedia.org/wiki/Microservices
- https://blog.karmawifi.com/how-we-build-microservices-at-karma-71497a89bfb4#.nz66s9wxk
- http://www.slideshare.net/InfoQ/from-a-monolith-to-microservices-rest-the-evolution-of-linkedins-service-architecture
- http://www.pwc.com/us/en/technology-forecast/2014/cloud-computing/features/microservices.html