VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMATIKOS STUDIJŲ PROGRAMA

Pagrindiniai komunikavimo integracijų mikroservisų architektūrose tipai ir jų analizė

Main communication integration types in microservices architectures comparison and analysis

Kursinis darbas

Atliko: Lukas Milašauskas (parašas)

Darbo vadovas: Dr. Saulius Minkevičius (parašas)

Vilnius – 2020

TURINYS

Įvadas

Temos aktualumas

Tobulėjant programinės įrangos (toliau PĮ) kūrimo įrankiams ir vis augant informacinių sistemų sudėtingumo poreikiams, kyla daug klausimų PĮ kūrėjams kokį architektūros modelį ir kokias technologijas pasirinkti, pradedant kurti naują informacinę sistemą. Vis daugėja skirtingų technologijų ir programavimo kalbų, kurios yra pranašesnes už kitas tik siaurose srityse, dėl to dažnu atveju neužtenka pasirinkti vieną technologiją ar programavimo kalbą norint sukurti kokybišką ir tvarią PĮ. Kai kurios technologijos yra pranašesnės resursų taupyme, kitos pranašesnės daug skirtingų bibliotekų palaikymu ir lengvai naudojama aplikacijų programavimo sąsają (angl. *„Application programming interface“* arba *„API“*) ir t.t. Kuriant naują PĮ reikia gerai išsianalizuoti tuo metu esamas technologijas ir jų privalumus. Įmonės yra linkusios kurti PĮ tokiomis technologijomis, kurių specialistų yra daug ir kurie norėtų palaikyti ir kurti jomis. Renkantis skirtingas technogolijas, atsiranda problema kaip jas apjungti, kad veiktų vieningai. Tokiu atveju, galima naudotis saitynų tarnybų pagalba (angl. ”Web services”), tačiau to neužtenka, nes kas, jeigu norime, skirtingus funkcionalumus įgyvendinti skirtingų technologijų pagalba. Tada dažnu atveju naudojamas mikroservisų (angl. *„Microservices“*) architektūrinis modelis. Remiantis šiuo modeliu būtų kuriami atskiri moduliai (taip pavadinsime atskirus sitemos vienetus), kurių kiekvienas būtų atsakingas už savo funkcionalumą. Problema su šia architektūra, kad reikia priversti šiuos skritingus modulius bendrauti tarpusavyje, o tai nebūna taip paprasta.

Problema

Kokį integracijų tipa rinktis mikroservisų architektūrose, norint įgyvendinti komunikaciją tarp skirtingų servisų.

Darbo tikslas

Palyginti skirtingus komunikacijų tipus, apžvelgti jū pranašumus ir trūkumus, pateikti technologijų šiem tipam realizuoti pvyzdžių.

Uždaviniai tikslui pasiekti

1. Remiantis literatūra apibūdinti, kas yra mikroservisai, kuo jie ypatingi, kaip jie projektuojami.
2. Išskirti pagrindinius integracijų ir komunikavimo tipus ir jų savybes.
3. Palyginti skirtingus komunikavimo būdus mikroservisų architektūrose.
4. Pateikti konkrečius komunikavimo integracijų ir technologijų pavyzdžius.
5. Pateikti rekomendacijas, kokiais atvejais, kokius tipus būtų geriau naudoti.

1. Kas yra mikroservisai

1.1. Monolitinės sistemos ir mikroservisų atsiradimas

Pagal autorių Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara publikuotą straipsnį „Microservices: yesterday, today, and tomorrow“ [DGL+17] jau 1960-iais buvo susiduriama su problemomis susijusiomis su dižiulio masto PĮ kūrimu ir kaip tai projektuoti. Buvo kuriama daug būdų kaip tai daryti, ir daug toerijų kaip tūrėtų atrodyti PĮ kodas, ir kaip projektuoti informacines sitemas. Daug veliau apie 2000 metus, susiformavo sąvokos „Service-Oriented Computing“ (toliau SOC) ir „Service-Oriented Architecture” (toliau SOA), kurių idėja buvo ir paradigmos buvo apie tai, kad aplikacija, arba kitaip pavadinus servisas, turėtų būti atsakingas už konkretaus resursų ar verslo logikos informaciją. Šį servisą tūri būti galima pasiekti su konkrečia technologija ir taip komunikuoti ir gauti informaciją. Taigi remiantis jau anksčiau minėtu straipsniu „Microservices: yesterday, today, and tomorrow“ [Mis7] iš SOC ir SOA kiek vėliau susiformavo mikroservisų idėja ir paradigmos. Pagal Martin Fowler ir James Lewis 2016-ais metais publikuotą straipsnį „Microservices“ [FL14] šis terminas mikroservisai buvo pirmą kartą diskutuotas 2011 metais, Venecijoje vykusiose PĮ kūrimo architektų dirbutvėse (angl. *„workshop“*). Po metų ta pati grupė architektų nusprendė, labiausiai tinkantis pavadinimas šiam architektūriniam tipui yra mikroservisai (angl. *„microservices“*). Po šiuo terminu slypi daug idėjų ir paradigmų, tačiau pagrindinė mintis yra skaidymas dideles sistemos į mažas „granules“ ir mažus servisus. Taip pat norint apibūdinti senas sistemas, kurios buvo nedalomos ir sistemą yra paleidžiamas kaip vienas vienetas, atsirado terminas „monolitas“ (angl. *„monolith“*). Nors šis terminas ir buvo naudojamas Unix bendruomenės jau ilgą laiką iki mikroservisų susikūrimo. Remiantis Eric Steven Raymon knyga „The Art of UNIX Programming“ [Ray03], kuri buvo išleista 2003 metais, terminas „monolitas“ buvo naudojamas apibūdinti sistemoms, kurios yra per didelės. Taigi, šie du terminai naudojami iki šios apibūdinti informacinių sistemų architektūriniams tipams.

1.2. Mikroservisų pranašumai prieš monolitus

Pagrindinius principus ir mikroservisų pranašumus detaliai išdėstė Sam Newman savo knygoje „Building Mircroservices: Designing Fine-Grained Systems“ [New15]. Šiame leidinyje autorius išgrynina keletą labai svarbių mirkoservisų privalumų.

1.2.1. Technologijų nevienalytiškumas (angl. ***„Technology Heterogeneity“***)

Kiekvienas servisas turi atlikti skirtingas funkcijas ir turėti skirtingas atsakomybes. Norint pasiekti geriausius rezultatus galima rinktis skirtingas technologijas, kurios būtų geriausiai pritaikytos konkrečiam uždaviniui spręsti. Todėl mikroservisuose kiekvieną servisa galim projektuoti skirtingom technologijom, pavyzdžiui skirtingomis programavimo kalbomis.

1.2.2. Atsparumas (angl. *„Resilience“*)

Atsitikus problemai ir sugriuvus vienai sistemos komponentei, monolitinėje sistemoje tektų perkraudinėti arba taisyti visą sistemą. Mikroservisų architektūroje stambių pasikeitimų būtų išvengta ir žlugtų tik vieno serviso veikimas. Tokiu atveju kiti servisai apie tai nežinotų ir veiktų toliau, o norint ištaisyti problemą, užtektų sutaisyti ir perkrauti vieną servisą.

1.2.3. Plečiamumas (angl. *„Scaling“*)

Stambioje monolitinėje sistemoje visos plečiamumo ir efektyvumo problemos spendžiamos kartu. Mikroservisų sistemoje kievieną tokio tipo problemą sprendžiame atskiruose servisuose. Tokiu atveju servisus, kurie reikalauja mažiau resursų galima suteikti mažiau resursų, o sunkesniems ir mažiau efektyviems servisams išskirti daugiau. Tačiau verta paminėti, kad mikroservisų sistemos plečiamumas ne visada yra geras, tai aprasšyta Omar Al-Debagy ir Peter Martinek straipsnyje „A Comparative Review of Microservices and Monolithic Architecture“ [AM18]

1.2.4. Lengvas diegimas (angl. *„Ease of Deployment“*)

Vystant PĮ dažnai susiduriama su diegimo problema. Su naujais funkcionalumais reikia iš naujo diegti naują PĮ versiją. Monolitinėje sistemoje tenka iš naujo sudiegti visą sistemą, tačiau mikroservisų sistemoje galima sudiegti tik tuos servisus, kurie yra susiję su pakeitimais.

1.2.5. Organizacinis pasiskirstymas (angl. *„Organizational Allignment“*)

Dažnai įmonėse prie informacinės sitemos dirba daug žmonių. Jie būna pasiskirstę komandomis ir turi skirtingas atsakomybes. Mikroservisų sistemose galima išvengti komunikavimo incidentų ir kiekvienai komandai dirbti su skirtingais servisais. Tokiu pavydžiu dirba stambi informacinių technologijų (toliau IT) įmonė „Netflix“.

1.2.6. Kompozicija (angl. *„Composability“*)

Įmonės dažnai susiduria su problema, kai tas pats funckionalumas reikalingas keliose informacinėse sistemose. Mikroservisų architektūroje, kadangi sistema susideda iš mažų autonomiškų servisų, juos galima atskirti ir perpanaudoti skirtingose sistemose, arba kitai sistemai suteikti prieigą tik prie konkrečių resursų, o ne visos sistemos.

1.2.7. Optimizuotas pakeičiamumas (angl. *„Optimizing for Replaceability“*)

Dirbant vidutinio dydžio arba didelėse imonėse dažnai susiduriama su problema, kai naudojama sena kodo bazė ir pasenusios technologijos. Dažnai tokią sistemą reikia atnaujinti siekiant efektyvumo arba palaikymo paprastumo. Tokiu atveju norint atnaujinti bibliotekas arba technologijas, tenka iš naujo perprogramuoti dalį sistemos. Mikroservisų architektūros pagalba, tai tampa žymiai papraščiau, kai užtenka perrašyti konkretų servisą, neliečiant likusios informacinės sitemos.

1.3. Monolitinių sistemų skaidymas į mikroservisus

Paskutinį dešimtmetį tapo gan populiaru stambaus mąsto monolitus skaidyti į mikroservisus, tačiau tai nėra taip paprasta. Visų pirma skaidant monolitą į atskirus servisus labai svarbu identifikuoti, kokie mažesni servisai bus. Serviso riboms apibrėžti panaudosime Micheal C. Feahters knygoje „Working Effectively with Legacy Code“ [Fea04] apibrėžtą terminą „siūle“ (angl. *„seam“*).Siūle šiuo atveju yra kodo dalis, kuri yra izoliuota ir autonomiška. Siūles ir bus mūsų atskiri mikroservisai. Autorė Susan J. Fowler savo knygoje „Production-Ready Microservices“ [Fow17] aprašė patarimus ir žingsnius, kaip reikėtų skaidyti monolitą į mikroservisus. Ji pamini, kad tai reikėtų daryti etapais:

1. Monolitinę aplikaciją paleisti su tiek kopijų kiek turėsime siūlių.
2. Išskirstyti kvietimus į kopijas, pagal tai kokias siūles kopijos reprezentuoja.
3. Išvalyti aplikacijos kopijas, paliekant tik siūlių funkcionalumą.

Verta paminėti, kad po kiekvieno išvardinto žingsnio bus atliekami testavimai, kurie patikrintų ar sistema veikia korektiškai.

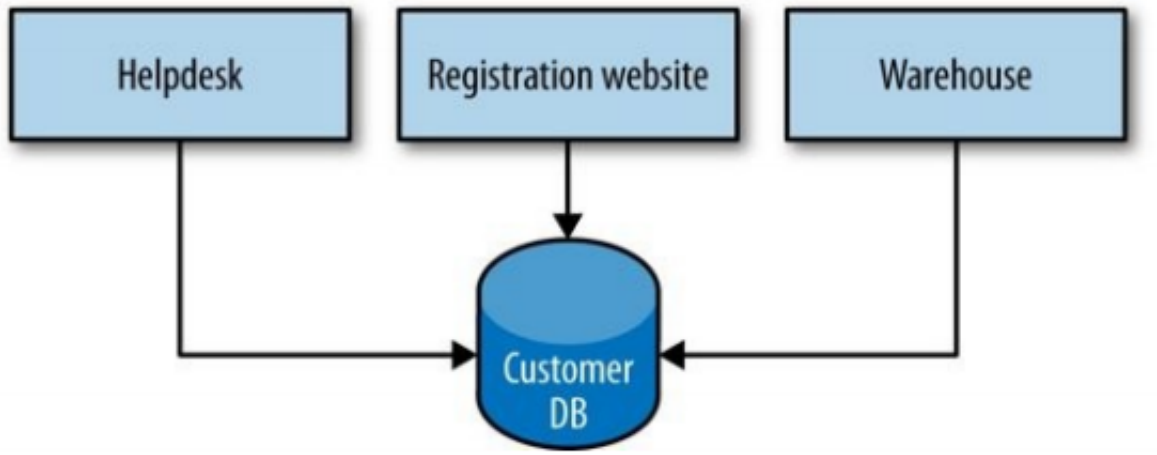
2. Mikroservisų sistemos vidinių integracijų tipai

Pagal Sam Newman knygą „Building Mircroservices: Designing Fine-Grained Systems“ [New15] vienas iš pagrindinių integracijų projektavimo apsektų yra stengtis išvengti nesuderinamų pakeitimų (angl. *„breaking changes“*). Norint išvengti tokių pakeitimų, reikia rinktis tokį technologinį sprendimą, kad pakeitus vieno mikroserviso grąžinamų duomenų struktūrą, kiti servisai be pakeitimų veiktų ir galėtų gauti duomenis. Sam Newman teigimu „tinkamos integracijos pasirinkimas yra pats svarbiausias technologinis su mikroservisais susijęs dalykas“. Todėl labai svarbu yra pasirinkti tinkamą integracijų tipą, nes nuo to priklauso kiek daug problemų sukels naujų funkcionalumų kūrimas. Autorius išskyria keletą skirtingų mikroservisų komunikavimo tipų, kuriuos ir aptarsime:

1. Servisų jungimas per duomenų bazę.
2. Sinchroninės užklausos/atsakymo (angl. *„request/response“*) integracijos.
3. Asinchroninės įvykiais paremtos (angl. *„event-based“*) integracijos.

2.1. Servisų komunikavimas per duomenų bazę

Šis integracinis tipas yra paremtas vienos bendros duomenų bazės naudojimu per keletą skirtingų mikroservisų. Šis tipas ypatingas tuo, kad kiekvienas mikroservisas turi priėjimą prie tų pačių resursų, tik modifikuoja resursus už juos atsakingi mikroservisai. Mikroservisų sistemos komunikavimo per duomenų bazę schema (1 pav.):



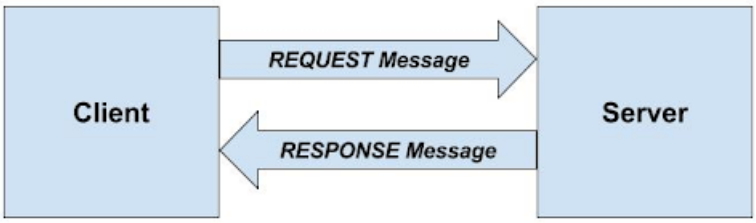
1 pav. Mikroservisų komunikavimas duomenų bazės pagalba.

Svarbu paminėti, jog toks komunikacijų tipas yra labai nesaugus ir duomenų migravimas kuriant stambesnius funkcionalumus yra neišvengiamas. Pačio Sam Newman, knygos [New15] autoriaus, nuomone, tai yra praščiausias komunikavimo būdas ir stipriai nerekomenduojamas. Dėl šios priežasties komunikavimo per bendrą duomenų bazę daugiau šiame darbe nenagrinėsime.

3. Sinchroninės (angl. “synchronous”) integracijos

3.1. Sinchroninių integracijų principas

Sinchroninės integracijos dar gali būti apibūdinamos kaip „dviejų žmonių komunikavimas realiu laiku“ [HKL09]. Su sinchroniniais komunikavimo pavyzdžiais dažnai susiduriame visi. Vienas iš tokių yra apsilankymas paprastoje internetinėje svetainėje. Interneto naršyklėje suvedus puslapio pavadinimą, HTTP protokolo pagalba, mums iš serverio yra užkraunamas svetainės turinys ir atvaizduojamas. Šis komunikavimo būdas yra paremtas užklausos/atsakymo (angl. *„request/response“*) principu. Šiame modelyje egzistuoja Klientas (tai yra mūsų naršyklė) ir serveris (tai yra interneto svetainės savininkas arba įmonė teikianti svetainių talpinimo paslaugas). Klientas siunčia užklausą serveriui prašydamas resurso, šiuo atveju tai yra mūsų norimos pamatyti svetainės turinio, ir laukia kol svetainė atiduos atsakymą į užklausą. Serveris reaguodamas į kliento užklausą grąžina atsakymą, kuris būti arba svetainės turinys, arba grąžinama klaida, pavyzdžiui kad toks resuras neegzistuoja arba klientas yra neautorizuotas šios svetainės lankytojas. Šis bendravimas yra perteiktas pateikta schema (2 pav.):



2 pav. Sinchroninio komunikavimo schema.

Taigi šis modelis puikiai gali veikti ir mikroservisų sistemoje. Vienas servisas siunčia užklausą į kitą servisą norėdamas gauti informaciją arba inicijuoti, kokį nors veiksmą. Šis modelis ypatingas tuo, kad yra primityvus ir iškart užklausą išsiuntęs servisas žinos ar sėkmingai pavyko atlikti norimą veiksmą. Šis modelis yra labai geras, kai tik įvykus sėkmingai užklausai leidžiame vartotojui vykdyti kitas operacijas ir kitaip bendrauti su mūsų sistema, tai yra vartotojo autentifikacija ir autorizacija.

3.2. Sinchroninių integracijų technologijų tipai

Būdų realizuoti sinchroninius komunikavimo modelius yra daug. Šiuo metu populiariausi yra du:

- RESTful saityno tarnybos.
- SOAP saityno tarnybos.

Turint omenyje, kad REST yra tik architektūrinis stilius, o ne protokolas, ir ju nelabai galima lyginti [Loi18]. Tačiau remiantis Joni Makkonen magistrinio darbo „REST ir SOAP saityno tarnybų efektyvumo ir naudojamumo palyginimas“ [Mak17] galima teigti, kad REST yra pranašesne ir šiame darbe išsiplėsime tik su šiuo architektūriniu stilium.

REST saityno tarnybos užklausos struktūra yra paprasta. Ji susideda iš keletos atributų:

- HTTP metodo, pvz.: *„GET“, „POST“, „PUT“, „DELETE“*.
- Unikalaus resurso identifikatoriaus (arba adreso), pvz.: *„http://mif.vu.lt/“*.
- Antraščių (angl. *„Headers“*), pvz.: *„Content-Type: application/json“*.
- Užklausos turinio (čia gali būti bet koks standartinis formatas, pvz.: *„JSON“*).
- HTTP protokolo versija, pvz.: *„HTTP/1.1“*.

REST atsakymo struktūra skiriasi nuo užklausos tik tuo, kad vietoje HTTP metodo, gaunamas HTTP statuso kodas. Tokio komunikavimo per REST saityno tarnybas pavyzdį galime pamatyti šioje schemoje (3 pav.):

3 pav. REST saityno tarnybų schema.

4. Asinchroninės (angl. “asynchronous”) integracijos

4.1. Asinchroninių integracijų principas

Asinchroninių integracijų principas

4.2. Asinchroninių integracijų technologijų tipai

Asinchroninių integracijų technologijų tipai

5. Skirtingų integracijų tipų privalumai ir trūkumai

Skirtingų integracijų tipų privalumai ir trūkumai

6. Rezultatai ir išvados

6.1. Rezultatai

Rezultatai

6.2. išvados

išvados

Šaltiniai

[AM18] Omar Al-Debagy and Peter Martinek. A comparative review of microservices and monolithic architectures. https://arxiv.org/pdf/1905.07997.pdf, 2018.

[DGL+17] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. Microservices: yesterday, today, and tomorrow. https://arxiv.org/pdf/1606.04036.pdf, 2017.

[Fea04] Micheal C. Feathers. *Working Effectively with Legacy Code*. Prentice Hall PTR, Upper Saddle River, United States of America, 2004. 29 psl.

[FL14] Martin Fowler and James Lewis. Microservices. https://martinfowler.com/articles/microservices.html, 2014.

[Fow17] Susan J. Fowler. *Production-Ready Microservices*. O'Reilly Media, Sebastopol, United States of America, 2017. 1 psl.

[HKL09] Volker Hockmann, Heinz D. Knoell ir Ernst L. Leiss. *Encyclopedia of Multimedia Technology and Networking, Second Edition*. 2009. 173 skyrius.

[Loi18] J. Loisel. Soap vs rest (why comparing them is a nonsense. https://octoperf.com/blog/2018/03/26/soap-vs-rest/, 2018.

[Mak17] Joni Makkonen. *Performance and usage comparison between REST and SOAP web services*. Magistrinis darbas, Aalto University, 2017.

[New15] Sam Newman. *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media, Sebastopol, United States of America, 2015. 39 psl.

[Ray03] Eric Steven Raymond. *The Art of UNIX Programming*. Addison-Wesley, Upper Saddle River, United States of America, 2003. 1 psl.

Sąvokų apibrėžimai

Sąvokų apibrėžimai ir santrumpų sąrašas sudaromas tada, kai darbo tekste vartojami specialūs paaiškinimo reikalaujantys terminai ir rečiau sutinkamos santrumpos.