TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Alo Martin Pallase

Monoliitse ärirakenduse kaasajastamine pilveteenuse abil metsaregistri näitel

Bakalaureusetöö (9 EAP)

Juhendaja(d): Margit Konno, Pelle Jakovits

Tartu 2024

Monoliitse ärirakenduse kaasajastamine pilveteenuse abil metsaregistri näitel

Lühikokkuvõte:

Mikroteenuste arhitektuur pakub paindlikke ja skaleeruvaid lahendusi traditsiooniliste monoliitsüsteemide piirangute ületamiseks. Integratsioon monoliitarhitektuurilt mikroteenustele on kasvav trend IT-sektoris, mis võimaldab ettevõtetel suurendada oma süsteemide paindlikkust ja hooldatavust. Mikroteenused võimaldavad rakendusi pilvekeskkonnas majutada, võimaldadades kiiremaid uuendusi ja hooldust. Käesolev töö käsitleb mikroteenustele ülemineku protsessi, tuues välja peamised raskused nagu andmete konsistentsi säilitamine ja süsteemi komponentide vaheline suhtlus. Samuti esitatakse lahendused ja soovitused nende raskuste ületamiseks. Kokkuvõttes näitab see bakalaureusetöö, kuidas mikroteenuste arhitektuurile üleminek toetab jätkusuutlikku IT arengut.

Võtmesõnad: mikroteenused, monoliit arhitektuur, pilveteenused

CERCS: P175 Informaatika, süsteemiteooria

Modernizing a Monolithic Business Application Using Cloud Services: A Case Study of the Estonian Forest Registry

Abstract:

Microservices architecture offers flexible and scalable solutions to overcome the limitations of traditional monolithic systems. The transition from monolithic architecture to microservices is a growing trend in the IT sector, allowing companies to increase the flexibility and maintainability of their systems. Microservices enable hosting applications in cloud environments, facilitating faster updates and maintenance. This work addresses the process of transitioning to microservices, highlighting major challenges such as maintaining data consistency and communication between system components. Solutions and recommendations for overcoming these challenges are also presented. In conclusion, this bachelor's thesis demonstrates how the transition to a microservices architecture supports sustainable IT development.

Keywords: microservices, monolithic architecture, cloud computing

CERCS: P175 Informatics, systems theory

Lühendid ja mõisted

Metsaregister	Metsaressursi arvestuse riiklik register
KeMIT	Keskkonnaministeeriumi Infotehnoloogiakeskus
RIHA	Riigi infosüsteemi haldussüsteem
SaaS	*Software as a service*
PaaS	*Platform as a service*
IaaS	*Infrastructure as a service*
FaaS	*Function as a Service*
REST	*Representational State Transfer*
JSON	*JavaScript Oject Notation*
DevOps	*Developement and operations*

Sisukord

Sissejuhatus

Pidevalt arenev infotehnoloogia valdkond toob kaasa uuenduse vajaduse igas infosüsteemis. Tehnoloogiad vananevad, leitakse turvaauke või kaob ära arendaja poolne tugi. Samuti liigutakse kõikjal protsesside, sh tarkvaraarenduse ja tarkvara tarnimise protsesside optimeerimise suunas, mis võimaldab tõsta tarkvaraarenduse ja operatiivtegevuste efektiivsust, kasumlikkust ja kvaliteeti. Seetõttu peavad organisatsioonid pidevalt hooldama ja uuendama oma tarkvara.

Monoliitsüsteemide pika kasutusajaloo tõttu on tänapäeval väga suuri rakendusi, mida on raske uuendada ning mille ülalpidamine nõuab palju ressursse ja suurt arendusmeeskonda. Selliste süsteemine hooldamine võib osutuda väga kulukaks [1]. Probleemi lahendamiseks on levinud viis lahutada rakendus funktsionaalsuste poolest eraldiseisvateks tükkideks ehk viia üle mikroteenuste arhitektuurile. Suurte süsteemide puhul võib olla keeruline viia rakendus ühe korraga uuele arhitektuurile, mistõttu tuleb teha seda väikeste sammude abil, et hoida olemasolev rakendus pidevalt toimimas.

Tarkvaraarenduse projektides on tihti puudujääk ajast. Selleks, et kulutada võimalikult vähe aega korduvate tegevuste tegemisele on kasutavad meeskonnad DevOps arendusmeetodit. Selle tulemusena automatiseeritakse koodi tarneprotsesse, testimist ja integratsiooni samme, jättes rohkem aega arendustööle.

Eelnevast lähtuvalt on **probleemiks:** 2017 aastal loodud metsaregistri infosüsteemi (vt ptk 2) monoliitsus ei võimalda pilveteenusele üleminekut ja DevOps põhimõtete rakendamist.

Eesmärk: Kaasajastada metsaregistri infosüsteemi läbi arhitektuurimuudatuste teostamise lähtuvalt kliendi poolt esitatud lähteülesandest.

Autori ülesandeks oli töökäigus tehtud uute rakenduste konteineriseerimine ja pilvevalmiduse tagamine ning DevOps põhimõtete juurutamine meeskonnas.

Hüpotees: metsaregistri monoliitne arhitektuur on võimalik teha pilvekeskkonnaga ühilduvaks.

Ülesanded, millised tuleb eesmärgi täitmiseks lahendada:

- Uurida teoreetilist tausta kaasaegsete tehnoloogiate kohta, millised võimaldavad pilveteenustele üleminekut;

- Tutvuda KliM haldusala tehnoloogilise profiili ja mittefunktsionaalsete nõuetega [2];
- Planeerida kliendi poolt esitatud lähteülesandest tulenevalt arhitektuurimuudatused;
- Realiseerida arhitektuuri muudatused, mis võimaldavad viia rakenduse üle pilveteenusele.

1 Tehnoloogiliste aluste ülevaade

1.1 Monoliitarhitektuur

Monoliitarhitektuur on laialdaselt kasutatav tarkvaraarenduse mudel, kus rakendused on ühendatud ühtseks süsteemiks. Tihti pakitakse süsteemi esirakendus ja tagarakendus ühte arhiivifaili, nagu peatükis 2.2 on kirjeldatud, mida mõne serveri abil käivitatakse.

Monoliitarhitektuur on olnud juba varajasest ajast kasutusel. Paljud tarkvara projektid arendatakse esialgu just sellise arhitektuuri alusel. 1960. aastatel kirjutatud esimestes arvutiprogrammides oli monoliitarhitektuuri kasutamine tavapärane [3]. Esmastel arvutitel oli tänaste arvutitega võrdlemiselt väike jõudlus, seega ka neile loodud programmid olid lihtsa ehitusega, mis on monoliit arhitektuuri suureks eeliseks. Väikeste rakenduste puhul on monoliitsüsteeme lihtne arendada, samas on sellised süsteemid ka väga efektiivsed, mida võib pidada üheks nende levimise põhjuseks.

Monoliitsete süsteemide ülesseadmise lihtsus, arenduse kiirus ja vigade leidmine on põhjus, miks mudelit kasutatakse tänapäevastes rakendustes (vt ptk 2.1). Rakenduse pidev funktsionaalsuse kasv hakkab lõpuks monoliit süsteemi koormama. Uute muudatuste lisamine ja rakenduse hooldamine muutub koodibaasi suurenedes järjest raskemaks [4].

1.2 Mikroteenused

Mikroteenused on üheks tuntumaks arhitektuuri mudeliks monoliidi kõrval. Mikroteenuste nimetus tuleneb nende struktuurist, mis koosneb väiksematest, iseseisvalt funktsioneerivatest teenustest. Mudelit võib pidada mõnes mõttes ka monoliit arhitektuuri vastandiks. Kui monoliit süsteemi iseloomustab rakenduste tihe seotus, siis mikroteenuste puhul on vastupidi, teenused on üksteisest sõltumatud.

Teiseks suureks erinevuseks mikroteenuste ja monoliit arhitektuuri vahel võib pidada ka nende majutamist. Mikroteenuseid saab majutada pilve keskkonnas, kus teenused on konteineriseeritud rakendused, mille näite leiab peatükist 3.3.2. Tänu teenuste madala sidususele on võimalik neid paindlikumalt hallata.

Monoliit süsteemilt mikroteenustele üleminekuks on erinevaid põhjuseid, näiteks eelnevalt mainitud koodibaasi kasv ja uute muudatuste lisamise keerukus, mida käsitleb peatükk 3.3.1. Üldiselt on mikroteenuste kasutusele võtt kasvavas trendis [5].

Süsteemi tükeldamiseks on vaja mingit alguspunkti. Lauretis [6] väidab, et üheks tükeldamise võimaluseks oleks vaadelda ja analüüsida süsteemi äriloogikat ja seejärel jagada funktsionaalsuse põhjal mikroteenusteks. Äriloogika põhjal jagamine ei ole ainus viis, veel on näiteks võimalik jagada arendaja kogemuse või hariduse põhjal, rakenduse teenuste (vt ptk 3.3.2) ja arhitektuuri stiili põhjal [7].

1.3 Pilvetehnoloogia ja Eesti Riigipilv

Kaasaegses infotehnoloogia maailmas, kus jõudlus, ressursside kohene saadavus ja kulude optimeerimine on kasvavas trendis, tuleb leida uuenduslikke lahendusi, mis pakuvad paindlikkust, skaleeritavust ja vahendite efektiivset kasutamist. Üheks selliseks tehnoloogiaks on pilveteenus, mis on muutnud organisatsioonide, ettevõtete ja individuaalsete kasutajate lähenemist andmetöötlusele ning IT-infrastruktuuri haldamisele.

Kuna käesoleval ajal on pilvetehnoloogia järjest kasvava tähtsusega valdkond ning IT infrastruktuuri arengus üks olulisemaid suundi [8], siis on ka Keskkonnaministeeriumi Infotehnoloogiakeskus (KeMIT) teinud valiku, et viib paari aasta jooksul enda hallatavad infosüsteemid üle Eesti Riigipilve. Metsaregistri infosüsteemi üleviimine riigipilve on kavandatud aastasse 2025 (vt ptk 3.1).

Riigipilv on riigi omandis olev pilv, milles on võimalik kasutada avaliku pilve teenuseid. Riigipilve alustaristu on:

- API juhitav
- iseteenindusega
- virtualiseeritud

„Lisaks valitsussektorile on Riigipilve kasutajad kohalikud omavalitsused, elutähtsa teenuse osutajad (ETO), riigile IT teenuseid pakkuvad eraettevõtted ja tervishoiuteenuse osutajad. [9]“

1.4 Konteinerite tehnoloogia

Tehnoloogia kiire areng toob kaasa paremate meetodite leiutamise ja rakendamise. Kasutajate pidevalt muutuvate vajadustega ja kiirete muutuste tempoga toimetulekuks peame muutma viisi, kuidas me muudatusi rakendame. Konteineriseerimine on tehnoloogia, mis aitab seda teha, võimaldades ehitada, levitada ja käivitada mis tahes rakendusi igal pool.

Tuntuimaks konteineriseerimise tehnoloogia arendajaks on Docker[1]. Konteiner tehnoloogial on mitmeid eeliseid tavapäraste virtuaalmasinatega võrreldes, nendest peamiseks on väiksem mälu vajadus ning kergem seadistus [10].

Konteinerite tehnoloogia on võimas, kuid suuremate süsteemide puhul, kus võib olla sadu ja rohkem konteinerit, muutub nende haldamine käsitsi äärmiselt keeruliseks ja aeganõudvaks, mistõttu on oluline kasutada automatiseeritud tööriistu ja süsteeme nende efektiivseks juhtimiseks. Kubernetes[2] on üheks levinumaks orkestreerimis tööriistaks on Google projektist edasi arenenud avatud lähtekoodiga tehnoloogia [11].

Dockeri ja Kubernetese kasutamisega on meil lahendatud peamised probleemid tarkvara kiireks arendamiseks. Endiselt, aga on vaja konteinerite tegemiseks anda juhiseid kasutades Dockeri käskude faili nimega Dockerfile, või andes käsklusi käsurealt. Selle protsessi automatiseerimiseks on otstarbekas kasutada mõnda integratsiooniserverit (vt ptk 3.4.4), mis selle töö ära teeb. Tuntumateks on Jenkins[3] ja Gitlab[4] serverid. Integratsiooni serveritega on võimalik kogu arendusprotsessi arendamisest kuni rakenduse tarneni luua üheks tervikuks. See on võimalik tänu konveieritele, kus saab määrata etapid iga sammu jaoks [12].

1.5 DevOps

Suurimaks ressursiks tänapäeval on aeg. DevOps on meetod, mis keskendub arendusprotsesside automatiseerimisel, eesmärgiga kiirendada tarkvaraarendust ning parandada koostööd meeskondades.

Automatiseerimise tulemusena väheneb arendaja töötundide arv, mis vähendab arenduskulusid ja suurendab efektiivsust tehes samad tööd ära kiiremini. Buttar et al. [13] saavutasid 60% väiksemad kulud kasutades DevOps meetodeid pilve rakenduses.

DevOps'i peamisteks eelisteks on kiirem tarkvaraarendus (vt ptk 3.4.4), väiksemad kulud ja suurenenud reageerimisvõime kliendi vajadustele, mis kõik aitavad kaasa ettevõtte üldisele

[1] https://www.docker.com/ (10.05.2024)

[2] https://kubernetes.io/ (15.05.2024)

[3] https://www.jenkins.io/ (15.05.2024)

[4] https://about.gitlab.com/ (15.05.2024)

edule ja innovatsioonivõimele. Samuti tasub mainida, et meetodi kasutamine aitab ettevõtetel püsida konkurentsis ning parandada tiimi sisest koostööd [14].

2 Metsaregistri infosüsteem

2.1 Metsaregistri infosüsteemi kirjeldus

Metsaressursi arvestuse riikliku registri põhimäärus kohaselt on metsaregistri pidamise eesmärk koguda ja säilitada andmeid metsa paiknemise, pindala, tagavara, kasutamise ja seisundi kohta [15]. Metsaregistris hoitakse metsa inventeerimisandmeid ning metsateatiste, keskkonnaametnike välitööde, metsauuendus- ja metsakaitseekspertiiside andmeid. Metsaregistri metsaportaal on mõeldud kasutamiseks avalikkusele Eesti metsade kohta info saamiseks ning metsakorraldajatele ja metsaomanikele riigiga suhtlemiseks. Metsaportaal võimaldab oma metsaga seotud asjaajamistel silma peal hoida ning esitada riigile taotlusi ja olla pidevalt kursis menetluse seisuga.

Infosüsteem loodi riigihange „Metsaressursi arvestuse riikliku registri terviklahenduse uue platvormi arendus ning migratsioon“ käigus ja võeti kasutusele 2017 aastal. Arendustöid on teostatud pidevalt, kuid vajalike suuremate arhitektuuriliste muudatuste tegemist hakati planeerima 2023 aastal, st enam kui viis aastat peale infosüsteemi kasutuselevõttu (vt ptk 3.3).

Metsaregister on ülesehituselt geograafiline infosüsteem (GIS), mis koosneb kolmest rakendusest: avalikkusele suunatud veebirakendus (https://register.metsad.ee/) ehk metsaportaal (AVE), ametnikele suunatud veebirakendus (AME) ning ametnike arvutitesse Software Centeri kaudu installeeritav välitöövahendi (VTV) võrguväline tarkvara. Metsaregistri arhitektuuri kirjeldab ptk 2.2.

2.2 Metsaregistri arhitektuur

Metsaregister koosneb kolmest eelnevas peatükis nimetatud rakendusest - avalik metsaportaal, ametniku portaal ja välitöö rakendus, nagu eelnevalt mainitud peatükis 2.1.

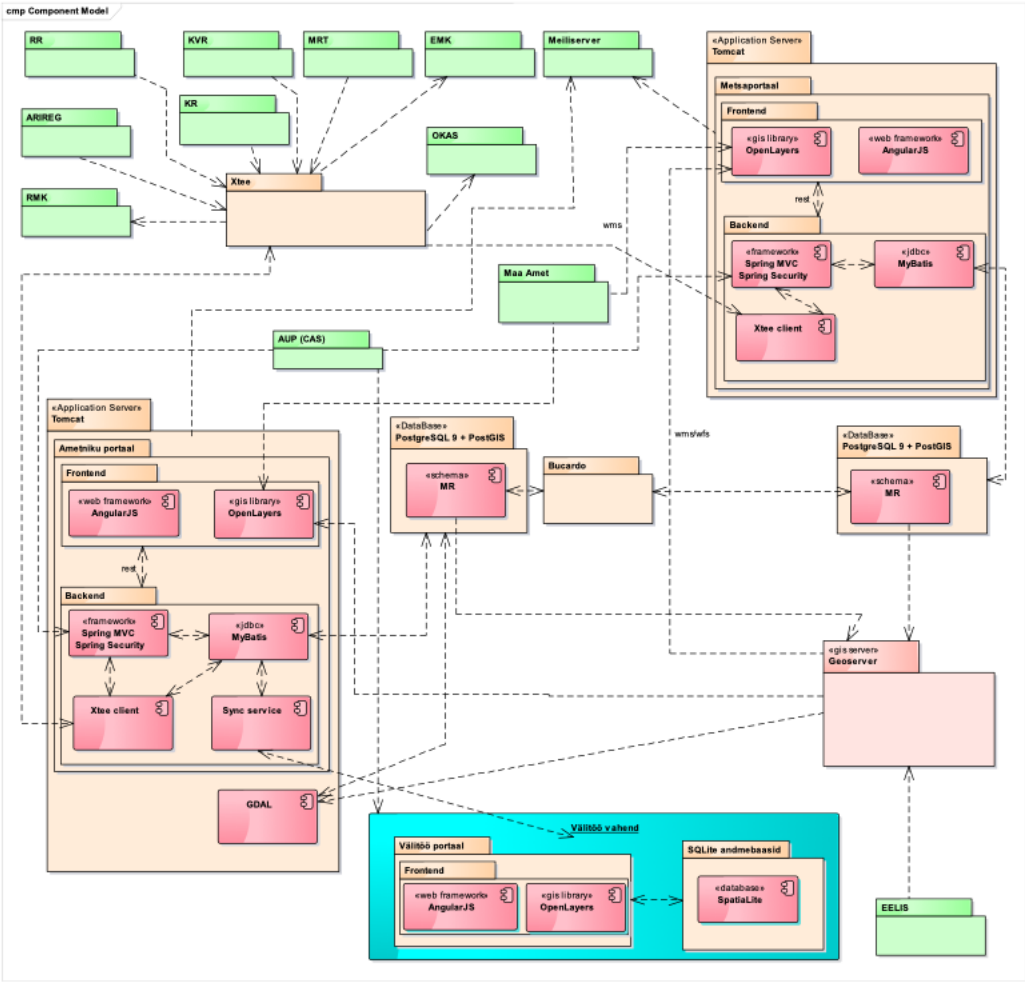
RIHA-st [16] leitava Metsaregistri arhitektuuridokumendi järgi on ametniku rakendus ja avaliku portaali rakendused sama arhitektuuriga ja kasutavad serveri poolel Spring MVC[5] raamistikku. Kasutajaliides on lahendatud AngularJS[6] raamistikuga. Rakenduste esitluskihid ja tagarakendused on ehitatud ühte WAR-faili, mis tähendab, et kogu rakenduse

[5] https://docs.spring.io/spring-framework/reference/web/webmvc.html (15.05.2024)

[6] https://angularjs.org/ (15.05.2024)

kood, sealhulgas nii esitluskiht kui ka tagarakendused, pakitakse kokku üheks arhiiviks, mida veebiserveris käitatakse. Brauseri ja serveri vaheline suhtlus toimub REST teenuste abil JSON formaadis.

Joonis 1 kujutab kasutusel on olevaid kahte virtuaalmasinat, üks majutab avaliku portaali rakendust ning teine ametniku rakendust. Rakendusi jooksutab Apache Tomcat 8[7], mis kasutab Java[8] versioon 8-t.



Joonis 1. Vana süsteemi füüsiline arhitektuur

[7] https://tomcat.apache.org/download-80.cgi (15.05.2024)

[8] https://www.oracle.com/java/technologies/java8.html (15.05.2024)

Rakendused suhtlevad PostgreSQL[9] andmebaasidega. Kasutusel on kaks baasi, üks ametniku rakenduse jaoks teine avaliku portaali jaoks. Andmebaaside omavahelist sünkroniseerimist teostab Bucardo[10].

Rakenduse failide salvestamine ja lugemine toimub kummagi serveri kettalt. Samuti hoiustatakse kasutaja poolt üleslaetud faile serveris.

Rakenduse kood paikneb ühes Gitlabi koodihoidlas ning koodi publitseerimiseks on valmistatud Jenkinsi konveierid.

Kuigi algselt oli monoliitne arhitektuur praktiline, ei sobi see hiljem riigipilve integreerimiseks. Metsaregistri arhitektuuri muudatusi kajastab järgnev peatükk 3.

[9] https://www.postgresql.org/ (15.05.2024)

[10] https://bucardo.org/ (15.05.2024)

3 Metsaregistri arhitektuuri muudatused

3.1 Arhitektuurimuudatuste teostamine kui projekt

Oma olemuselt on metsaregistri infosüsteemi arhitektuurimuudatuste teostamine IT-projekt oma kindla alguse, lõpu (vt ptk **Error! Reference source not found.**) ja eelarvega, millised o n ühtlasi ka projekti edukuse põhikriteeriumiteks.

Enamasti koosneb iga projekt järgmistest etappidest [17]:

- Algatamine
- Planeerimine
- Teostamine
- Lõpetamine

3.2 Projekti algatamine

Projekti sisu määratleti arhitektuurimuudatuste teostamise lähteülesandega. Lähteülesanne arhitektuurimuudatuste teostamiseks esitati vastava hanke dokumentide seas ja tööde teostamiseks sõlmiti hankeleping 2023 aasta aprillis. Tööde valmise tähtajaks planeeriti 28.08.2023.

Ülesandeks oli metsaregistri arhitektuuri kaasajastamine:

1. Viia ametniku ja avaliku rakenduse andmebaasid üheks ja kaotades ära andmebaaside omavahelist sünkroniseerimist teostav Bucardo
2. viia rakendused Kubernetesese keskkonda, et valmis olla Eesti Riigipilve kasutuselevõtuks.

Valmiv lahendus peab vastama KeMIT-i tehnoloogilisele profiilile [2]. Viimane versioon on leitav KeMIT-i veebilehelt[11].

Lähteülesanne oli lakooniline ja ei sisaldanud detaile.

[11] https://www.kemit.ee/et/tehnoloogia (15.05.2024)

3.3 Planeerimine

3.3.1 Arhitektuurimuudatuste planeerimine

Projekti edukaks rakendamiseks oli lähteülesande detailide vähesuse tõttu hädavajalik koostöös tellijaga läbi viia põhjalik planeerimise protsess. Planeerimise etapp keskendus mitte ainult esmaste eesmärkide täpsustamisele, vaid ka nende saavutamiseks vajalike sammude selgitamisele. Tööde ettevalmistamisel arvestati nii tehnoloogilisi uuendusi kui ka olemasoleva süsteemi ümberkujundamist, et tagada riigipilve ülemineku võimalus ning vastavus KeMIT-i tehnoloogilisele profiilile.

Kuna tehnoloogiate valimine ja arhitektuurilised otsused jäävad rakendust pikalt mõjutama, et kasulikud mõjud oleksid maksimaalsed, tuleb nende osas valikuid teadlikult teha. Planeerimise käigus kinnitati tehnoloogiate valik ja arhitektuuriotsused ning koostati detailne tööplaan, et täpselt määratleda tegevuste järjekord, vajalikud ressursid ning võimalikud riskid, tagades nii projekti nõuetele vastavuse ja süsteemi tõhusa toimimise.

Metsaregistri süsteemi tehnoloogiate valikul oli kitsenduseks kliendi tehnoloogiline profiil, mida lähteülesande kirjeldamisel põgusalt mainiti. Ülejäänud tehnoloogiad, mida töös planeeriti kasutada, st mis ei olnud kliendi poolt määratud, valiti võttes arvesse tehnoloogia populaarsust ja arendaja enda teadmisi või alternatiivse tehnoloogia puudumise tõttu.

Planeeritud tegevused sisaldasid muuhulgas eraldiseisvate esirakenduste ja tagarakenduste loomist, süsteemi üleviimist Java ja Tomcat lahendustelt Spring Boot peale ning rakenduste paigutamist eraldi konteineritesse. Samuti kavandati põhjalikud lokaalsed ja Kubernetese keskkonnas toimuvad testid, et tagada süsteemi tõrgeteta toimimine pärast ümberkorraldusi.

Metsaregistri rakendusele on lisatud funktsionaalsust 2017-aastast nagu on öeldud ptk 2.1. See on muutnud olemasoleva koodibaasi ja rakendused suureks. Mikroteenuste üheks valiku põhjuseks võib välja tuua eelnevalt mainitud monoliit arhitektuuri puuduseid suurte rakenduste puhul, mida kirjeldab ptk 1.1.

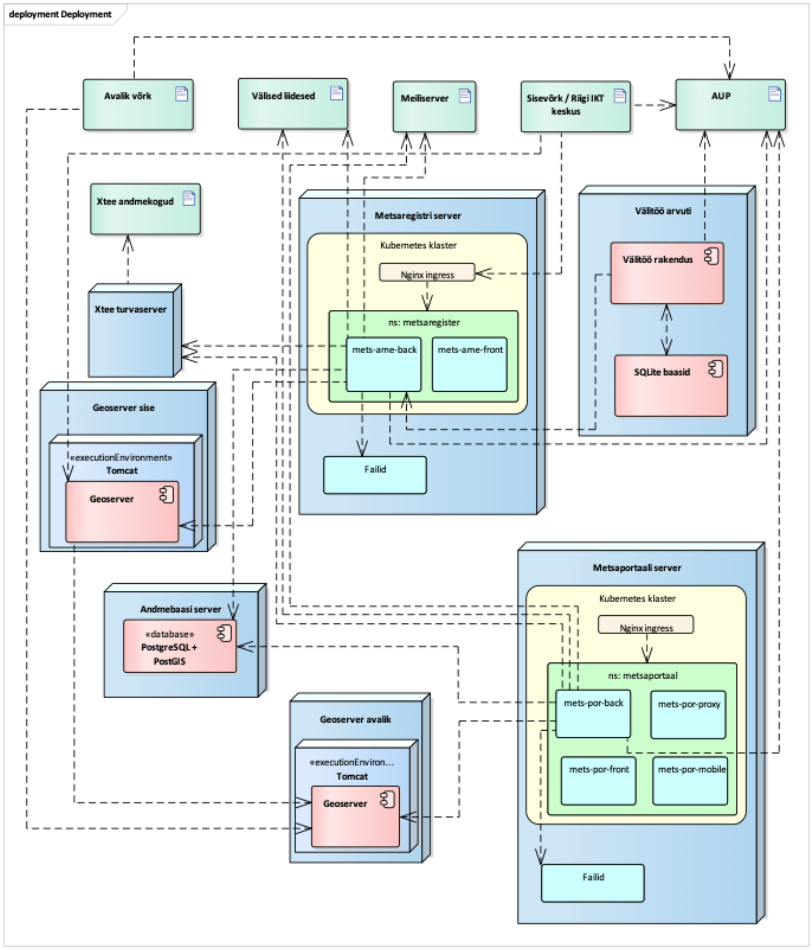
Teiseks suureks kitsenduseks otsuste tegemisel oli projekti fikseeritud ajaraamistik (vt ptk 3.2), mis piiras rakenduste tükeldamist väiksemateks teenusteks ning esirakendustes uuema Angular[12] raamistiku kasutusele võttu.

[12] https://angular.io/ (04.04.2024)

3.3.2 Planeeritavad arhitektuurimuudatused

Pilve kasutusele võtmisel on tavapärane liikuda mikroteenuste arhitektuurile mitmel põhjusel, sellest rääkis ptk 1.2. Mikroteenuste kasutusele võtmiseks on vajalik vana süsteemi arhitektuurist (vt ptk 2.2) tekitada üksteisest sõltumatuid teenuseid.

Tükkide tegemiseks on mitmeid võimalusi, mis on välja toodud ptk 1.2 lõpus. Algne plaan on tükid teha ametniku ja avaliku portaali rakendustest. Teenuste tegemiseks on vaja rakenduste esitluskiht ja funktsionaalne kiht WAR-failist välja tuua ning tekitada nendest 2 eraldi seisvat rakendust. Rakenduste eraldamisel oleks võimalik kumbki panna eraldi konteineritesse, tekitades kokku neli erinevat teenust nagu on näha järgneval Joonis 2.



Joonis 2. Metsaregistri uue planeeritud arhitektuuri paigaldusvaade

Tekkinud teenused on võimalik viia pilvekeskkonda. Samuti on igat teenust selle tulemusena lihtsam hooldada, ühe teenuse koodi muudatusel ei pea kogu süsteemi uuendama. Failide kättesaamise, mis varasemalt toimus otse serverist küsides ja sinna salvestades, saab lahendada Kubernetese mahu haakimise objekti ingl k *volume mount* abil,

mis loob tunneli konteineri virtuaalkeskkonna ja süsteemi serveri vahel kus faile hoiustatakse.

3.4 Realiseerimine

3.4.1 Andmebaasid

Andmebaaside osas toimus projektis oluline ümberkorraldus, mille peamiseks eesmärgiks oli vähendada andmete dubleerimist ja sellega seotud sünkroniseerimisprobleeme. Varem kasutusel olnud kahe andmebaasi süsteem tekitas mitmeid operatiivseid tõrkeid, eriti kuna Bucardo, mida kasutasime andmete sünkroniseerimiseks, osutus sageli ebausaldusväärseks. See tõi kaasa sagedased süsteemi katkestused ja raskendas andmete ajakohasena hoidmist. Lisaks, kui oli tarvis teha muudatusi andmestruktuurides, nõudis see muudatuste tegemist mõlemas andmebaasis, mis võttis kaua aega ja suurendas vea tegemise võimalust.

Andmebaaside ühendamiseks koostasime integratsiooni skripti, mis kopeeris puuduolevad kirjed avaliku portaali andmebaasist ametniku andmebaasi. Skript korrastas ka kirjete järjestused ühendatud andmebaasis õigeks ning eemaldas Bucardo päästikud. Andmebaasile lisasime rollipõhised kasutajad erinevate õigustega tabelitele ja kirjetele, et piirata ametniku rakenduse kirjete ligipääsu avalikule portaali rakendusele.

3.4.2 Spring Boot

Tagarakendused said töö käigus suure refaktoreerimise. Varasemalt kasutusel olnud *Java ja* Tomcati pealt viisime rakendused Spring Boot versioon 4.22 peale. Muudatustega saavutasime projektki kiirema käivitumise. Vana arhitektuuriga võttis tagarakenduse käivitamine aega ligikaudu 20 sekundit. Uue lahenduse puhul võtab see keskmiselt aega 3 sekundit. Samuti paranes rakenduse pakkimisele kuluv aeg, mis on enamjaolt mõjutatud esirakenduse eraldamisest.

3.4.3 Docker ja Kubernetes

Valisime Dockeri konteinerite loomiseks ning Kubernetese nende orkestreerimiseks. Iga rakenduse jaoks tekitasime Dockerfile-i, nagu on näha Lisa 1*,* kus on kirjeldatud konteineri loomise reeglid alustades tõmmisfaili täpsustamisest jätkates valmis rakenduse kood kopeerimisega ja lõpetades rakenduse käivitamise käsuga. Faili jooksutamisel tehtaks samm haaval kõik ette antud käsud ning valmib vastav konteiner virtuaal keskkonnaga.

Rakenduse pakkimine ja ehitamine on samuti võimalik teha sama Dockerfile sees, kuid otsustasime seda teha Gitlabi toruga, et hoida valmiv rakenduse tõmmisfail võimalikult väiksena. Samuti tuli ajaliselt kiirem teha rakenduse ehitamise protsess Gitlabi toru sammuna.

Igal projektil on eraldi Kubernetese konfiguratsioonifailid, kus peamised parameetrid on muutujatena nagu on näha Lisa 2. Muutujad väärtustatakse Gitlabi keskkonnas rakenduse paigaldamise sammul.

3.4.4 Gitlab konveierid

DevOps meetodit, mida käsitlesime lõigus 1.5, kasutades saab valminud teenuseid kiiremini tarnida. Ajaliselt võttis esialgne lahendus rakenduse muudatuse ülessurumisest koodihoidlasse kuni testkeskkonnas uue versiooni avaldumiseni aega keskmiselt 6 minutit. Uute teenustega on see aeg peaaegu kolm korda kiirem. Kõige suurema koodibaasiga ametniku tagarakendus jõuab nüüd testkeskkonda umbes 2 minutiga. Samuti on võimalik kõiki rakendusi paralleelselt uuendada ilma, et peaks kogu süsteemi uuesti ehitama.

Vana arhitektuuriga toimusid tarned Jenkinsi konveierite abil. Otsustasime asendada need kliendi poolt majutatud Gitlabi konveierite vastu ning kasutada Artifactory[13]-t rakenduse pakkide hoiustamiseks.

Konveierid on üles seatud toimetama rakendused arendus-, testimis- ja tootmiskeskkonda, mis pakub pidevintegratsiooni ja tagab rakenduse pidevvalmiduse. Muudatuse üles lükkamisel Gitlabi koodibaasi käivitub automaatselt toru, mis viib läbi testid, ehitab rakenduse ja pakib selle konteinerisse ning seejärel paigutab konteinerid testkeskkonda, kus testijad saavad kontrollida muudatused üle. Tootmiskeskkonda paigaldamine jäi manuaalseks, et vältida ekslike tarneid tootmiskeskkonda.

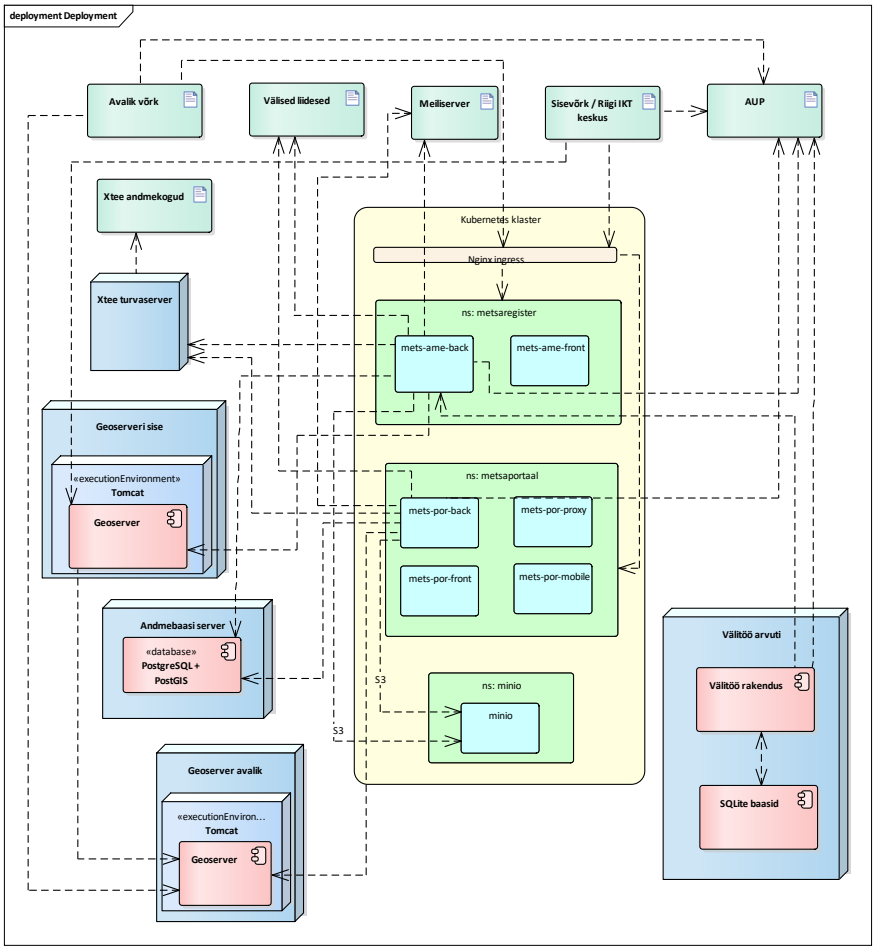
Iga teenus on nüüd eraldi repositooriumis ning kõigile on tehtud eraldi skriptid torude jooksutamiseks, mille näite leiab Lisa 3-st. Uue koodi projekti üles surumisel käivitub skript ja tekitab rakendusest uue versiooni. Nagu eelnevalt mainitud, on igas projektis eraldi Dockerfile ja Kubernetese konfiguratsiooni failid. Torudel on kolm etappi, esiteks ehitatakse rakendus Gitlabi virtuaalkeskkonnas valmis. Seejärel rakendatakse Dockerfile-st leitavad juhendid ja tehakse rakendusest uus konteiner ning viimaks rakendatakse

[13] https://jfrog.com/artifactory/ (04.05.2024)

Kubernetese konfiguratsiooni faile, kus väärtustatakse eelnevalt mainitud muutujate väärtused Gitlabi keskkonnas seadistavate muutujate väärtustega.

3.4.5 Valminud arhitektuuri terviklahendus

Lõplik arhitektuurilahendus koosnes kuuest eraldi konteinerist nagu on näha Joonis 3. Ametniku rakendus ja avaliku portaali rakendus paiknevad eraldi nimeruumides, välist ühendust kummalegi rakenduse teenustele vahendab Kubernetese ingress[14] objekt, mis suunab kasutajaid õigele rakendusele.



Joonis 3. Metsaregistri uue arhitektuuri paigaldusvaade

Planeeritud arhitektuurist erineb valmis lahendus peamiselt minio[15] objekthoidla kasutusele võtmise poolest. See muudatus sai kliendi soovil tehtud, et muuta failide hoiustamine samuti

[14] https://kubernetes.io/docs/concepts/services-networking/ingress/ (10.05.2024)

[15] https://min.io/ (01.05.2024)

pilvepõhiseks. Meie ülesandeks oli tagarakendused S3 protokolliga ühilduvaks teha. Klient andis ette minio teenuse otspunkti kust andmeid pärida.

3.5 Projekti lõpetamine ja analüüs

3.5.1 Projekti edutegurid ja ebaõnnestumise põhjused

Suure monoliitse arhitektuuriga süsteemi arhitektuurimuudatuste teostamine ei ole lihtne ülesanne. Töö käigus võib tekkida mitmeid probleeme ja takistusi.

Järgnevad loetelud ja lõik tugineb Hendrik Ekke Altnurme magistritööle [18].

Autor on välja toonud, et klassikalisi IT projekti edukriteeriumeid on kolm:

- olemine ajaraamis,
- olemine eelarves
- omamine kokkulepitud tarkvaralist funktsionaalsust.

Lisaks võiks projekti , või ebaõnnestumist kaardistades järgida veel kolme kriteeriumit:

- IT lahendusest saadav kasu kliendile,
- lahenduse tehniline kvaliteet (töökindlus, hooldatavus ja muud mitte-funktsionaalsed omadused),
- projektitöö efektiivsus.

IT projekti nimetamine ebaõnnestunuks on üldjuhul subjektiivne. Näiteks projekt läks üle eelarve, lahendus jõudis kasutajateni plaanitust hiljem või väljendus ebaõnnestumine töö madalas kvaliteedis. Lihtsustatult võib öelda, et ebaõnnestunud IT projekt on üldjuhul projekt, mis ei täida sellele seatud eesmärke.

Autor on kasutanud projekti analüüsimiseks lihtsustatud retrospektiivi läbiviimise meetodit, mis põhineb agiilses tarkvaraarenduses kasutataval Deming tsükli mudelil PDCA (Plan-Do-Check-Act) [19]. Selle järgi analüüsitakse:

- Plussid ehk mis läks projektis hästi?
- Miinused ehk milliste probleemidega puutus meeskond projekti käigus kokku?
- Ideed ehk milliseid uusi ideid saadi selle retrospektiivi käigus?
- Plaan ehk milliseid muudatusettepanekuid on järgmiseks projektiks?

Eneseanalüüsiks kasutas autor STARR [20] meetodit.

3.5.2 Projekti analüüs

Metsaregistri infosüsteemi arhitektuurimuudatuste projekti puhul peab autor projekti õnnestunuks, kuna lähteülesande eesmärgid said täidetud.

Plussid – mis läks projektis hästi:

- Planeeritud tööde teostamiseks ettevõttes leiduva kompetentsi olemasolu
- Ühtehoidev, üksteist abistav ja kohusetundlik meeskond
- Ajaraamis püsimiseks võimalus meeskonda liikmeid lisada
- Lähteülesandes kirjeldatud tööde teostamine

Miinused – probleemid millega meeskond kokku puutus:

- Lähteülesande ebapiisav detailsus
- Puudujäägid planeerimise protsessis – ei suudetud kõiki kliendi soove välja selgitada, mistõttu tuli mõningad tehtud tööd ümber teha ja uuesti testida
- Kliendi poolel tehnilise juhi vahetus – esialgsed kokkulepped ja soovid muutusid
- Töö käigus lisandunud planeerimata tööd (nt minio)
- Meeskonna vähene kogemus mõningate tööde teostamisel
- Testimisele kulunud suur ajakulu tulenevalt automaattestide vähesusest
- Integratsioonitestide mittetoimimine - projekti täitmiseks ette nähtud aeg oli piiratud ja integratsioonitestide parandamise vajadust ei olnud alguses arvestatud, siis kahjuks projekti käigus nende taastamiseni ei jõutud
- Puudujäägid projektijuhtimises – ei osatud koostada realistlikku projektiplaani ega sellest kinni pidada ja puudusid kokkulepped detailsemate tegevuste teostamise kohta
- Tulenevalt täiendavatest töödest ja tegevustest, millised lähteülesandes ei kajastunud ja mida planeerimisel ei käsitletud, lõppes projekt planeeritust hiljem
- Tulenevalt sellest, projekt väljus planeeritavast ajaraamist, ei püsitud eelarves

Ideed – mida teha edaspidi paremini:

- Testimise ajakulu vähendamiseks peaks kirjutama rohkem automaatteste
- Kliendiga enne projekti arendamist teha kindlaks soovid ja need kirja panna

Plaanid – millised on metsaregistri edasiarendamise võimalused:

- Lahendust on võimalik tõsta riigipilve
- Valminud teenuste tükeldamine veel väiksemaks
- Uusi teenuseid saab arendada mikroteenustena

3.5.3 Eneseanalüüs

Eneseanalüüs STARR meetodil:

S (olukord) – metsaregistri arhitektuuri kaasajastmine.

T (ülesanne) – minu ülesandeks oli rakenduste konteineriseerimine ja orkestreerimine ning DevOps meetodi rakendamine, sealhulgas Gitlabi konveierite tegemine.

A (tegevused) – koostasin Dockeri konteinerid uutele rakendustele. Tegin Kubernetese konteinerite orkestreerimise skriptid ja paigaldasin süsteemi testkeskkonda. Täiendasin ja koostasin uusi Gitlabi konveierite skripte.

R (tulemus) – teostatud tööd on tänaseks kliendile üle antud ja toodangukeskkonda paigaldatud.

R (peegeldus) – Õppisin väga palju pilvetehnoloogia, Kubernetese ja Dockeri rakendamist. Sain kogemuse DevOps insenerina ja teha süsteemi arhitekti töid. Lisaks sain teadmisi geoinfosüsteemide arendamisest.

Kokkuvõte

Bakalaureusetöö raames teostati metsaregistri arhitektuurimuudatused, millised ja need on toodangukeskkonnas alates 3.mai 2024. Seega sai töö bakalaureusetöö eesmärk täidetud.

Eesmärgi täitmiseks lahendati mitmed ülesanded. Enne arhitektuuriööde teostamist analüüsiti erinevaid allikaid kaasaegsete tehnoloogiate kohta ja kuulati teostatava ettevõtte erinevate projektide meeskonnaliikmete seniseid kogemusi Kubernetese kasutuselevõtu osas, mis andis hea lähtekoha tööde teostamiseks. Samuti tutvuti uusimate KliM haldusala tehnoloogilise profiili ja mittefunktsionaalsete nõuetega. Planeeritud arhitektuurimuudatused teostati.

Hüpotees täitus, valmis süsteem, mida on võimalik pilvekeskkonnas kasutada.

Projekti õnnestumise analüüsimeetodina kasutati lihtsustatud retrospektiivi läbiviimise meetodit. Projekti analüüsist selgus, et meeskond puutus projekti käigus kokku paljude probleemidega. Vaatamata sellele, et klassikalistest IT projekti edukriteeriumitest on kaks ei osutunud edukaks (aeg ja raha), võib projekti sellegipoolest lugeda õnnestunuks, kuna lähteülesandes toodud eesmärgid täideti.

Kokkuvõtvalt on käesoleva töö peamised tulemused järgmised:

- Teostatud Metsaregistri arhitektuurimuudatused on toodangukeskkonnas alates 3. maist 2024
- Arhitektuuridokument sai uuendatud ning on loodetavasti peatselt kättesaadav ka RIHA-s
- Projekti teostajad, sh töö autor said suurepärase kogemuse arhitektuurimuudatuste teostamise projektis – said juurde palju uusi teadmisi ja praktilisi oskusi.

Töö tulem on aluseks järgnevate metsaregistri arendustööde teostamisel. Kokkuvõtte tegemise hetkeks on selgunud, et kliendil on testimisel juba uus funktsionaalsus, milline oli võimalik edukalt arendada mikroteenusena tänu uuenenud arhitektuurile. Metsaregistri uus arhitektuur võimaldab metsaregistri infosüsteemi viia vajadusel Eesti Riigipilve. Järgnevate arendustööde käigus on võimalik tükeldada olemasolevat süsteemi väikemateks mikroteenusteks ja luua uusi funktsionaalsusi mikroteenustena.

Lisaks sellele on kõigil, kes kavandavad olemasoleva infosüsteemi arhitektuurimuudatusi, võimalik saada mõningane ülevaade arhitektuuriga seonduvast temaatikast ja õppida projekti analüüsis välja toodud kogemustest.

Bakalaureusetöö käigus täideti ka teostaja isiklikud eesmärgid. Arenesid töö teostaja erialased teadmised ja oskused ning informatsiooni töötlemise ja kasutamise oskused. Autor sai ka palju uusi teadmisi geoinfosüsteemide arhitektuurist.

Viidatud kirjandus

[1] Bennett K., „Software evolution: past, present and future,“ *Information and Software Technology,* nr 38, pp. 673-680, 1996.

[2] KeMIT, „Tehnoloogia | KeMIT,“ [Võrgumaterjal]. Available: https://kemit.ee/et/tehnoloogia. [Kasutatud 05 05 2024].

[3] Xu KH, Zhang J, Gao S, „Froglingo-A Monolithic Alternative to DBMS, Programming Language, Web Server and File System,“ *ENASE,* kd. 1, pp. 247-252, 28 Nov 2010.

[4] T. J., „Microservices,“ *IEEE Software,* kd. 32, nr 1, pp. 116-116, 2015.

[5] Pahl C., Jamshidi P., „Microservices: A Systematic Mapping Study,“ London, 2016.

[6] L. L. De, „From Monolithic Architecture to Microservices Architecture,“ *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW),* pp. 93-96, 2019.

[7] Fritzsch J., Bogner J., Zimmermann A., Wagner S, „From Monolith to Microservices: A Classification of Refactoring Approaches,“ 2019.

[8] Majandus- ja Kommunikatsiooniministeerium, „Eesti riigipilve kontseptsioon,“ 2015. [Võrgumaterjal]. Available: https://riigipilv.ee/files/Eesti_riigipilve_kontseptsioon.pdf. [Kasutatud 10 05 2024].

[9] Eesti Riigipilv, „Riigipilvest,“ [Võrgumaterjal]. Available: https://www.riigipilv.ee/riigipilvest. [Kasutatud 10 05 2024].

[10] Docker Inc, „What is a container? | Docker,“ Docker Inc., [Võrgumaterjal]. Available: https://www.docker.com/resources/what-container/. [Kasutatud 10 05 2024].

[11] IBM, „The history of Kubernetes | IBM Blog,“ [Võrgumaterjal]. Available: https://www.ibm.com/blog/kubernetes-history/. [Kasutatud 10 05 2024].

[12] Red Hat Inc, „What is a CI/CD pipeline?,“ [Võrgumaterjal]. Available: https://www.redhat.com/en/topics/devops/what-cicd-pipeline. [Kasutatud 10 05 2024].

[13] Buttar A.M., Khalid A., Alenezi M., Akbar M. A., Rafi S., Gumaei A. H., Riaz M. T. , „Optimization of DevOps Transformation for Cloud-Based Applications,“ *Electronics,* kd. 12, nr 2, p. 357, 2023.

[14] Plant, O. H., Jos van Hillegersberg, Aldea A. , „How DevOps capabilities leverage firm competitive advantage: A systematic review of empirical evidence,“ *IEEE 23rd Conference on Business Informatics,* kd. 1, pp. 141-150, 2021.

[15] Riigi Teataja I, „Metsaressursi arvestuse riikliku registri põhimäärus,“ 16 06 2013. [Võrgumaterjal]. Available: Metsaressursi arvestuse riikliku registri põhimäärus. [Kasutatud 10 05 2024].

[16] RIHA, „Metsaressursi arvetuse riiklik register,“ [Võrgumaterjal]. Available: https://www.riha.ee/Infos%C3%BCsteemid/Vaata/metsaregister. [Kasutatud 10 05 2024].

[17] K. Eby, „Demystifying the 5 Phases of Project Management Try Smartsheet for Free By Kate Eby | May 29, 2018 (updated December 27, 2023) Share on Facebook Share on X Share on LinkedIn Copy link At the root of any successful project is a project manager (PM) worth,“ 29 Mai 2018. [Võrgumaterjal]. Available: https://www.smartsheet.com/blog/demystifying-5-phases-project-management. [Kasutatud 15 Mai 2024].

[18] Altnurme H. E., „Ebaõnnestumise juurpõhjused IT projektides ja nende vältimine,“ 2021.

[19] Bogdanov V., „How to conduct an effective retrospective in your Agile Team,“ 17 08 2015. [Võrgumaterjal]. Available: https://medium.com/@Intersog/how-to-conduct-an-effective-retrospective-in-your-agile-team-d5a022f2eba9. [Kasutatud 15 05 2024].

[20] Tartu Ülikool, „STARR Method for Self-evaluation of the Internship,“ [Võrgumaterjal]. Available: https://sisu.ut.ee/skytteinternship/starr-method-self-evaluation-traineeship/. [Kasutatud 15 05 2024].

Lisad

Lisa 1. Dockerfile näide

```
FROM eclipse-temurin:17

WORKDIR /var/app

RUN mkdir files
RUN mkdir properties

COPY ./mr_portaal/target/*.war ./app.war
COPY logback.xml /var/app/properties/
COPY application.properties /var/app/properties/
COPY mr-portaal.properties /var/app/properties/
COPY cacerts /var/app/

ENV JAVA_OPTS=""
ENV CATALINA_BASE=.
ENV TZ=Europe/Tallinn

ENTRYPOINT ["sh", "-c", "java ${JAVA_OPTS} -jar /var/app/app.war"]
```

Lisa 2. Kubernetese deployment.yaml faili näide

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ${CI_PROJECT_NAME}
  namespace: ${NAMESPACE}
  labels:
    app: ${CI_PROJECT_NAME}
    containerCI: ${CI_PROJECT_NAME}
    projectCI: ${CI_PROJECT_NAME}
    namespaceCI: ${NAMESPACE}
    relatedService: ${CMDB_CODE}
    projectName: ${CI_PROJECT_TITLE}
    environment: ${CI_ENVIRONMENT_NAME}
    servicecode: ${CMDB_CODE}
spec:
  replicas: 1
  revisionHistoryLimit: 2
  minReadySeconds: 25
  strategy:
    rollingUpdate:
      maxUnavailable: 0
      maxSurge: 1
  selector:
    matchLabels:
      app: ${CI_PROJECT_NAME}
  template:
    metadata:
      annotations:
        elastic.co/namespace: ${ELASTIC_LOG}
      labels:
        app: ${CI_PROJECT_NAME}
        environment: ${CI_ENVIRONMENT_NAME}
        servicecode: ${CMDB_CODE}
    spec:
      serviceAccountName: ${CMDB_CODE}
      securityContext:
        runAsNonRoot: true
      containers:
        - name: ${CI_PROJECT_NAME}
          image: ${HARBOR_REGISTRY}/${CMDB_CODE}/${CI_PROJECT_NAME}:$CI_COMMIT_REF_SLUG-$CI_COMMIT_SHORT_SHA
          imagePullPolicy: Always
          securityContext:
            runAsUser: 101
            runAsGroup: 101
            allowPrivilegeEscalation: false
          env:
            - JAVA_OPTS = ${JAVA_OPTS}
            - DB_HOST = ${DB_HOST}
            - DB_USER = ${DB_USER}
            - DB_PASS = ${DB_PASS}
            - AUP_URL = ${AUP_URL}
            - CI_ENVIRONMENT_URL = ${CI_ENVIRONMENT_URL}
            - LOG_LEVEL = ${LOG_LEVEL}
            - MID_REST_NAME = ${MID_REST_NAME}
            - MID_REST_URL = ${MID_REST_URL}
            - MID_TEST_ENV = ${MID_TEST_ENV}
            - MID_UUID = ${MID_UUID}
            - SMARTID_REST_NAME = ${SMARTID_REST_NAME}
            - SMARTID_URL = ${SMARTID_URL}
            - SMARTID_UUID = ${SMARTID_UUID}
            - SMTP_FROM = ${SMTP_FROM}
            - SMTP_HOST = ${SMTP_HOST}
            - SMTP_PORT = ${SMTP_PORT}
            - XTEE_instance = ${XTEE_instance}
            - XTEE_URL = ${XTEE_URL}
            - S3_ENABLED = ${S3_ENABLED}
            - S3_BUCKET = ${S3_BUCKET}
            - S3_HOST = ${S3_HOST}
            - S3_ACCESS_KEY = ${S3_ACCESS_KEY}
            - S3_SECRET_KEY = ${S3_SECRET_KEY}
            - CADASTER_LAYER_ID = ${CADASTER_LAYER_ID}
            - COPY_TEGEVUSTE_LOG = ${COPY_TEGEVUSTE_LOG}
            - EMAIL_WHITELIST = ${EMAIL_WHITELIST}
            - ERALDIS_KA_ID = ${ERALDIS_KA_ID}
            - RR_KONTROLL_SEES = ${RR_KONTROLL_SEES}
            - name: TEATIS_KA_ID
              value: "${TEATIS_KA_ID}"
            - RR_SOAP_CLIENT_TEST_CODE = ${RR_SOAP_CLIENT_TEST_CODE}
            - mtrMemberClass_override = ${mtrMemberClass_override}
            - mtrMemberCode_override = ${mtrMemberCode_override}
          resources:
            requests:
              memory: 256Mi
              cpu: 250m
```

Lisa 3. CI/CD skripti näide

Gitlabi CI/CD skripti fail muutujatega.

```
.deploy:
  stage: deploy
  only:
    - main
    - devel
    - tags
  image: harbor.sise.envir.ee/tools/kubectl-helm-helmfile:v1.19.5-v3.4.0-v0.135.0
  before_script:
  - unset KUBECONFIG
  script:
  - printenv
  - export IMAGE_PULL_SECRET=$CI_PROJECT_NAME
  - kubectl -n ${NAMESPACE} get secret ${CMDB_CODE} || kubectl -n ${NAMESPACE} create secret docker-registry ${CMDB_CODE} --docker-server=$HARBOR_REGISTRY
  - envsubst < ./manifests/serviceAccount.yaml | kubectl apply -f -
  - envsubst < ./manifests/hpa.yaml | kubectl apply -f -
  - envsubst < ./manifests/deployment.yaml | kubectl apply -f -
  - envsubst < ./manifests/service.yaml | kubectl apply -f -
  - envsubst < ./manifests/ingress.yaml | kubectl apply -f -

deploy-dev:
  extends: .deploy
  environment:
    name: dev
  only:
    - devel
  except:
  - tags
  tags:
    - kemit-apps-test-1
  when: manual
  variables:
    NAMESPACE: metsaregister-dev
    REPLICAS: 1
    MIN_PODS: 1
    MAX_PODS: 5
    CPU_LIMIT: 2000m
    MEMORY_LIMIT: 1024Mi
    CPU_REQUEST: 1000m
    MEMORY_REQUEST: 256Mi
    JAVA_OPTS: -Xmx512m -Xms512m

deploy-test:
  extends: .deploy
  environment:
    name: test
  tags:
  - kemit-apps-test-1
  only:
  - tags
  when: manual
  variables:
    NAMESPACE: metsaregister-test
    REPLICAS: 1
    MIN_PODS: 1
    MAX_PODS: 5
    CPU_LIMIT: 1500m
    MEMORY_LIMIT: 1024Mi
    CPU_REQUEST: 500m
    MEMORY_REQUEST: 256Mi
    JAVA_OPTS:  -Xmx512m -Xms512m


deploy-live:
  extends: .deploy
  environment:
    name: live
  tags:
  - kemit-apps-live-1
  only:
  - tags
  when: manual
  variables:
    NAMESPACE: metsaregister-live
    REPLICAS: 1
    MIN_PODS: 1
    MAX_PODS: 5
    CPU_LIMIT: 1500m
    MEMORY_LIMIT: 1024Mi
    CPU_REQUEST: 500m
    MEMORY_REQUEST: 256Mi
    JAVA_OPTS:  -Xmx512m -Xms512m
```

Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, Alo Martin Pallase,

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose, „Monoliitse ärirakenduse kaasajastamine pilveteenuse abil metsaregistri näitel“, mille juhendajad on Margit Konno ja Pelle Jakovits, reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.

2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 4.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.

3. Olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile.

4. Kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

Alo Martin Pallase

15.05.2024