TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Kert Kukk 174059IDDR

Deklaratiivset tarkvarajuurutamist abistava tööriista analüüs ja arendus

Diplomitöö

Juhendaja: Kristiina Hakk
PhD
Kaasjuhendaja: Rein Remmel
Rakenduskõrgharidus

Tallinn 2021

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Kert Kukk

17.05.2021

Annotatsioon

Käesolev töö lahendab probleemi, kus tarkvara juurutamisprotsessidest puudus ühtne tööriist, mis aitaks deklareerida rakenduste soovitud olekuid, ehk CI/CD (*Continuous integration/continuous delivery*) ahelast puudus deklaratiivse tarkvarajuurutamise praktikaid järgiv käsurea tööriist, mis vahetult enne juurutusprotsesse deklareeriks automaatselt rakenduste soovitud olekud.

Kirjatöö eesmärk on leida töös käsitletavale probleemile lahendus. Selle jaoks viiakse läbi juhtumiuuring, kus juhtumiuuringu objektiks on mitmed üksikjuhtumid. Kogutud andmete ja informatsiooni põhjal tehakse järeldused ja teostatakse lahenduse analüüs. Viimase abil kirjeldatakse probleemi lahendava tööriista nõuded, pärast mida alustatakse lahenduse planeerimise ja realiseerimisega. Lõpetuseks hinnatakse arendatud lahendust ja tehakse vastavad järeldused.

Lõputöö käigus arendati lahendus, mis hoolitseb selle eest, et deklaratiivsel tarkvarajuurutamisel tekkinud inimese poolt tehtavad veaohtlikud toimingud oleksid automatiseeritud ja ohutud. Selle tulemusena valmis vabavaraline käsurea tööriist, mis lahendas töös kirjeldatud probleemi ja on kasutatav erinevates tarkvarajuurutus keskkondades.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 27 leheküljel, 7 peatükki, 5 joonist, 15 tabelit.

Abstract

Analysis and Development of Declarative Software Deployment Helper Tool

The present thesis addresses the problem where software deployment processes did not have a tool that could help them to declare the desired states of the applications. In other words CI/CD (Continuous integration/continuous delivery) pipelines did not have a command line utility that could help them to declare the desired states of the applications before the deployment step. The purpose of this thesis is to find a solution to just described problem.

Multiple case studies were conducted to study the problem. For that reason interviews and code analysis were performed as a data collection method. After data collection analysis of the tool was made and functional and nonfunctional requirements were described. Then more technical part of the thesis began. Started with solution planning and moved to the development of the tool. Finally the developed solution was evaluated and corresponding conclusions were drawn.

As a result of the thesis, a command-line tool was developed. The purpose of the tool is to ensure that human error-prone actions during declarative software deployments are automated and safe. The final solution answers to the problem described in the thesis and because it is an open source project then it can be easily used in various CI/CD environments

The thesis is in Estonian and contains 27 pages of text, 7 chapters, 5 figures, 15 tables.

Lühendite ja mõistete sõnastik

Argo CD	Deklaratiivne GitOps Kubernetesele
AWK	Tekstifailide töötlemise keel
CD	*Contionus Delivery*, pidev juurutamine/tarne
CI	*Continous Intgration*, pidev integreerimine
DevOps	*Development and Operations*, arendus ja haldus
Docker	Arvutiprogramm, mis teostab operatsioonisüsteemi tasemel virtualiseerimist
Git	Vabavaraline hajutatud versioonihaldustarkvara
GitOps	Deklaratiivsel infrastruktuuril versioonidega hallatud CI/CD
Go	Teise nimega Golang on avatud lähtekoodiga programmeerimiskeel
Grep	Käsurea tööriist teksti otsimiseks kasutades regulaaravaldisi
Kubectl	Käsurea tööriist Kubernetese klastri kontrollimiseks
Kubernetes	Avaliku lähtekoodiga konteinerite haldussüsteem
Perl	Üldotstarbeline programmerimiskeel, mis loodi algselt teksti manipuleerimiseks
SED	Reale orienteeritud tekstitöötlusutiliit
SSH	*Secure Shell*, krüptograafiline võrguprotokoll turvaliseks võrguteenuste opereerimiseks turvamata võrgu kaudu
StrictHostKeyChecking	Märksõna kontrollimaks sisselogimisi masinatele, mille võõrustaja võti pole teada või on muutunud
YAML	*Yet Another Markup Language*, Inimesele lihtsasti loetav andmevahetuskeel

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Tehnoloogia areng on loonud uued võimalused kuidas arendada ja juurutada tarkvara. Kaasaaegsetes tarkvaraarendusettevõtetes, kus kasutatakse DevOps praktikaid soovivad arendusmeeskonnad, et nad saaksid võimalikult lihtsalt ja iseseisvalt tegeleda tarkvaraloomega. Sealjuures teha seda nii, et koodi ja tulemi tarne kliendile toimuks võimalikult kiiresti, turvaliselt ja usaldusväärselt, tõhustades sellega ettevõtte tegevus [1]. DevOpsi üks lähenemisviis on kasutada GitOpsi ehk praktikat, kus rakenduste soovitud olekud saavutatakse kasutades deklaratiivseid vahendeid. GitOps eemaldab paljud manuaalsetest toimingutest tulenevad veaohtlikud tegevused ja probleemid automatiseerides juurutusprotsessi [2]. Küll aga ei lahenda GitOps rakenduste soovitud olekute deklareerimise protsessi.

Diplomitöö teema valiku ajendiks oli Entigo OÜ poolt defineeritud probleem, kus erinevate projektide üleselt kasutati tarkvara juurutamisel GitOpsi ehk deklaratiivsel infrastruktuuril versioonidega hallatud CI/CD-d (*Continous Intgration/Contionus Delivery*), kuid vastava praktika ellu viimisel puudus ühtne tööriist.

Kirjeldatud probleemist tulenevalt on lõputöö eesmärk uurida ja analüüsida varasemaid lahendusi ning saadud teabe põhjal arendada uus tööriist, mis vastaks kogutud nõuetele ning asendaks vanad lahendused.

Lõputöö tulemusena valmib korduvkasutatav käsurea tööriist, mis on võimeline asendama erinevates projektides kasutusel olnud sarnaseid lahendusi. Uus tööriist hoolitseb selle eest, et deklaratiivsel tarkvarajuurutamisel tekkinud inimese poolt tehtavad veaohtlikud toimingud oleksid automatiseeritud, vähendades sellega ohtu, kus Kubernetese klastrisse jõuab soovimatu rakenduse olek.

Antud töö kasusaajateks on Entigo OÜ ja selle kliendid, ning potentsiaalselt kõik kes kasutavad deklaratiivse tarkvarajuurutamise juures GitOpsi.

Diplomitöö jaotus on järgnev:

- Probleemi kirjeldus ja töö eesmärgi jaotis annab informatsiooni ja taustainfot diplomitöös käsitletava teema kohta. Sõnastatakse töö probleem ja eesmärk ning tuuakse välja millist metoodikat probleemi lahendamiseks kasutatakse.

- Juhtumite analüüsi peatükis teostatakse juhtumiuuring, kus juhtumiuuringu objektiks on mitmed üksikjuhtumid. Peatükis kirjeldatakse kuidas toimus nende läbiviimine ja analüüsimine ning millistele järeldustele jõuti.

- Lahenduse analüüsis tuginetakse juhtumite analüüsi käigus kogutud andmetele ja järeldustele, mille abil kirjeldatakse nõuded lõputöö tulemusena valmivale tööriistale.

- Tehnilise teostuse peatükis kirjeldatakse kuidas planeeriti tehniline lahendus ja kuidas toimus selle realiseerimine.

- Tulemuste peatükis hinnatakse kas töös kasutatud uurimismeetod täitis oma eesmärki ehk andis piisava sisendi tööriista ehitamiseks ja kas arendatud utiliit vastas kirjeldatud nõuetele ning aitas lahendada töös kirjeldatud probleemi.

- Kokkuvõte sisaldab informatsiooni selle kohta mida antud töö käigus saavutati ja milliste tulemusteni jõuti.

2 Probleemi kirjeldus ja töö eesmärk

Selles peatükis antakse taustainfot lõputöö kohta, tuues välja diplomitöö teemaga seonduvad praktikad. Kirjeldatakse kirjatöö probleem ja eesmärk ning tuuakse välja millist metoodikat probleemi lahendamiseks kasutatakse.

2.1 Taustainfo

Lõputöö teema paremaks mõistmiseks mõtestatakse lahti, millises keskkonnas antud lõputöö probleem asetseb, alustades sellest kust tuleb termin deklaratiivne tarkvarajuurutus ja mida see tähendab.

2009. Aastal kasutas Patric Debois esimest korda Belgias toimunud DevOpsDays [5] konverentsi raames terminit DevOps. Seda võib nimetada DevOps praktika sünniajaks. Küll aga ei tähenda see seda, et see tekkis üleöö, vastupidi paljud inimesed ja liikumised on selle arengule kaasa rääkinud.

Deklaratiivne tarkvarajuurutus on DevOpsi üks osa. DevOps on kultuuriline liikumine, mis muudab seda kuidas inimesed mõtlevad oma tööst. See on midagi kuidas indiviidid mõõdavad sotsiaalse ja tehnilise muutuse mõju. See on mõtte- ja tööviis, mis võimaldab inimestel ja organisatsioonidel arendada ja säilitada jätkusuutlikke töövõtteid [3]. Olgu need siis kas tehnilise või mittetehnilise loomuga. DevOpsi põhiidee on tuua erinevad rühmad üksteisele lähemale kasutades automatiseeritud protsesse [4]. Töö kirjutamise ajaks on praktikast välja kujunenud parim tava mida paljud ettevõtted kasutavad ja mille poole paljude veel pürgivad [4], [6].

DevOps puudutab kõiki tarkvara arenduse ja operatsioonidega seotud etappe, alates planeerimisest ja arendamisest kuni monitoorimise ja paigalduseni tuues kokku oskused, protsessid ja tööriistad. Üheks DevOpsi nurgakiviks on pidev integratsioon ja pidev tarnimine ehk CI/CD, mis väga üldistatult tähendab automaatset koodi ehitamist, testimist ja juurutamist [6].

Pilvandmetöötlusvaldkonnas on CI/CD ja automaatne tarkvara paigaldamine ülimalt tähtis. Seda laadi keskkondades saab tarkvara paigaldamisele läheneda fundamentaalselt kahel erineval viisil - imperatiivselt ja deklaratiivselt. Imperatiivse tarkvarapaigalduse korral kasutatakse protseduurilise mudeleid, mis määravad sõnaselgelt täidetava

protsessi. Deklaratiivse tarkvarajuurutuse korral saavutatakse rakenduse struktuur ja olek kasutades erinevaid struktuurseid mudeleid, mille abil see realiseeritakse konkreetses paigalduskeskkonnas. Siinjuures on oluline märkida, et soovitud tulemuse elluviimise eest vastutab paigaldustarkvara [7]. See tähendab seda, et konfiguratsioon tagatakse faktidena, mitte juhistena. Paigaldustarkvara hoolitseb ise selle eest, kuidas kirjeldatud fakt täita, mitte ei hakata järgima konkreetseid juhiseid, kuidas minigi olekuni jõuda [8].

Lõputöö probleemist lähtuvalt tegeletakse käesoleva töö raames just deklaratiivse tarkvarapaigaldusega. Deklaratiivsete tööriistadega on võimalik kogu konfiguratsioon hoida Gitis, kus viimane toimib kui *single source of truth* (tõe allikas) [8]. See loob võimaluse Infrastructure as Code (infrastruktuurile kui koodile), lühidalt IaC, kus tervet infrastruktuuri hoiustatakse koodina modelleerides see skriptidesse. See omakorda võimaldab seda lihtsa vaevaga dubleerida teistesse keskkondadesse ilma, et kogu lahendus tuleks manuaalselt uuesti üles ehitada. Sedaviisi lähenemine on kergesti loetav ja hallatav võrreldes manuaalse ehitamisega [9]. Töö kirjutamise ajaks on pilvetehnoloogia kasutamine ja kasutuselevõtt läbiv isegi suurimate ja konservatiivsemate organisatsioonide seas [10]. Selline laialdane pilvetehnoloogiate kasutuselevõtt ja soov protsesse paremini automatiseerida on suuresti toimunud tänu Kubernetesele.

Kubernetes on portatiivne, laiendatav, vabavaraline platvorm konteinerite töövoo ja teenuste haldamiseks, sealjuures hõlbustades deklaratiivset konfigureerimist ja automatiseerimist [11]. Kuna Kubernetes on peaaegu täielikult deklaratiivne tööriist siis see on võimaldanud laiendada seda, kuidas hallata rakendusi ja nende operatsioonisüsteeme. Võimekus hallata ja võrrelda infrastruktuuri ja rakenduse olekut läbi Giti toimingute Gitis on rajanud tee GitOps filosoofiale ja tema praktikatele [8]. GitOps on viis kuidas rakendada pidevat tarnet pilvepõhistele rakendustele. Selle põhiidee seisneb selles, et Giti repositooriumis hoitakse deklaratiivne kirjeldus soovitud rakenduse või infrastruktuuri olekust, mis omakorda viiakse ellu automatiseeritud protsesside poolt [12].

GitOps praktika üks arvestatavamaid rakendajaid on Argo CD, mis on deklaratiivne pideva tarne tööriist Kubernetesele. Argo CD automatiseerib rakenduse oleku Gitis kirjeldatud olekuga vastavusse viimise [13]. Küll aga tuleb enne rakenduste juurutamist soovitud olek kuidagi ära defineerida. Seda saab teha kas manuaalselt või siis

automaatselt. Teise variandi korral peab selle jaoks olema masinliides. Kuna enne töö kirjutamist sellist ei leidunud, siis sellest tulenevalt tekkis vajadus see ehitada.

2.2 Probleem

Lõputöös lahendatav probleem pärineb Entigo OÜ-lt, kus erinevate projektide üleselt puudus tarkvara juurutamisprotsessidest ühtne tööriist, mis aitaks deklareerida rakenduste soovitud olekuid. CI/CD ahelast puudus GitOps praktikaid järgiv käsurea utiliit, mis vahetult enne juurutusprotsesse deklareeriks automaatselt rakenduste soovitud olekud.

2.3 Eesmärk

Tulenevalt eelnevalt kirjeldatud probleemist on töö eesmärk leida sellele selline lahendus, mis vastaks kirjeldatud probleemile ja oleks taaskasutatav lahendus tulevaste sarnaste probleemist tingitud situatsioonide korral. Selline lahendus võimaldaks korduvkasutamist ja vigade vältimist. Lahenduse realiseerimiseks teostatakse analüüs, et kirjeldada millistele nõuetele ja vajadustele lõpplahendus vastama peab. Töö raames uuritakse ja analüüsitakse kasutusel olevaid lahendusi ning saadud teabe põhjal arendatakse uus tööriist, mis vastaks kogutud nõuetele, oleks võimeline asendama uuritud lahendused ning oleks taaskasutatav tulevastes projektides nii Entigo OÜ, kui väliste kasutajate poolt.

2.4 Metoodika

Lõputöö probleemist ja eesmärkidest lähtuvalt kasutatakse töö läbiviimiseks metoodikat, mis koosneb analüüsist, tehnilisest realisatsioonist ja lahenduse hindamisest.

Analüüsi osa koosneb kahest suuremast peatükist. Esimeses peatükis juhindutakse juhtumiuuringu, kui empiirilise uurimismetoodika läbiviimise protsessist [14, ptk. 2.2], kus juhtumiuuringu objektiks on mitmed üksikjuhtumid. Protsessi läbiviimise käigus teostatakse probleemi lahendamisega seotud oleva kirjanduse uurimine, andmete kogumine ja andmete analüüsimine. Analüüsi teises osas tuginetakse esimese osa tulemile, mille abil sõnastatakse nõuded ja ootused uuele lahendusele. Analüüsile järgneb tehniline teostus, kus planeeritakse, kavandatakse ja realiseeritakse tehniline lahendus. Viimaseks leiab aset tulemuse hindamine ja testimine.

Töö metoodika valikul lähtutakse sellest, mis aitaks kõige paremini töös kirjeldatud eesmärke täita. Enne lõputöö tulemina valmiva tööriista ehitamist on vaja täpselt aru saada kus ja kuidas see probleemi lahendama hakkab ning millistele tingimustele ja nõuetele see vastama peab. Selle jaoks on vaja koguda ja analüüsida andmeid. Andmete kogumiseks ja analüüsimise uuritakse lõputöö probleemiga kokku puutunud projekte. Nende paremaks mõistmiseks teostatakse intervjuud ja koodianalüüsid, mis ühtlasi on abiks tööriista nõuete kirjeldamisel. Sellest tulenevalt kajastab autor igat töös vaadeldavat projekti, kus lõputöös käsitletav probleem aset leidis, kui eraldi juhtumiuuringu üksikjuhtumit. Selline lähenemisviis annab hea arusaama nähtusest ja selle kontekstist [14, ptk. 2.4], [15], mille analüüsi põhjal on võimalik kirjeldada tööriista ehitamiseks vajaminevad nõuded.

3 Juhtumite analüüs

Käesolevas peatükis teostatakse juhtumiuuring, kus juhtumiuuringu objektiks on mitmed üksikjuhtumid. Üksikjuhtumiteks on lõputöös kirjeldatud probleemiga kokku puutunud projektid. Peatükis kirjeldatakse kuidas viidi läbi juhtumiuuringud ja nende analüüsimine ning millistele järeldustele jõuti. Kogutud informatsioon loob eeldused lahenduse nõuete defineerimiseks, mida kajastab lahenduse analüüsis.

3.1 Juhtumiuuring

Juhtumiuuring, inglise keeles *case study*, on detailne uurimus mingi kindla teema kohta [15], kus juhtum on tegelikkuse analoog ehk otsene kogemus mistahes tegevusest [16]. Juhtumiuuringud sobivad hästi uuritava probleemi erinevate aspektide kirjeldamiseks, võrdlemiseks ja arusaamiseks [15].

Juhtumiuuringu objektiks on üksikjuhtum või mitmed üksikjuhtumid [18]. Mitme juhtumi uurimine annab nähtuse kohta rohkem informatsiooni. Protsessi läbiviimises on mõningad sarnasused katsetega – peamine mõte on saada täiendavat informatsiooni. Siiski ei tohiks seda segamini ajada statistilise replikatsiooni ja statistilise valimiga. Juhtumiuuringud tuginevad juhtumitele ja nende omadustele, olgu need siis mõnes mõttes kas tüüpilised või erilised [14, ptk. 7.5.1]. Lisaks tasub mainida, et juhtumiuuringu koostamine hõlmab endas tavaliselt kvalitatiivseid ja harvem ka kvantitatiivseid meetodeid [14, ptk. 2.3.2]. Juhtumiuuring võib samuti sisaldada teisi uurimismeetodeid ja vastupidi võib ise olla osa suuremast uuringust [14, ptk. 6.1].

Tarkvaratööstuses on töö jaotatud projektidesse. See annab võimaluse mõõta tarkvaraprojektide edukust ja ebaõnnestumist. Seetõttu on tarkvaraprojektid ilmselged kandidaadid juhtumiuuringuteks. Kuna kogu tarkvaraprojekti uurimine pikema aja jooksul on äärmiselt keeruline, siis tavaliselt kiputakse juhtumiuuringu raames uurima projekti mingit konkreetset aspekti või aspekte [14, ptk. 3.2.3]. Sarnaselt lähenetakse ka käesoleva töö probleemi uurimisele ehk juhtumiuuringu üksikjuhtumiteks on lõputöös käsitletava probleemiga kokku puutuvad tarkvaraprojektide juurutusetapid.

3.2 Juhtumiuuringu läbiviimise protsess

Enne juhtumiuuringu läbiviimist tutvuti ja uuriti erinevaid materjale, mis aitaksid mõista kuidas juhtumiuuringuid läbi viia. Autor juhindub antud töös kõige enam Hösti ja teiste kirjeldatud protsessist [14, ptk. 2.6], kuna seal kirjeldatud lähenemine sobib autori hinnangul kõige paremini käesoleva töö raames läbi viidava juhtumiuuringu teostamiseks.

Juhindudes viidatud protsessile peab juhtumiuuringut läbi viies arvestama viie etapiga:

1. juhtumi disain;
2. andmete kogumise ettevalmistus;
3. andmete kogumine;
4. andmete analüüsimine;
5. aruandlus.

Esimeses etapis kirjeldatakse juhtumiuuringu eesmärk ja läbiviimise plaan. Teises etapis defineeritakse andmete kogumise protseduur. Kolmandas etapis toimub andmete kogumine. Neljandas etapis toimub andmete analüüsimine ja viimases etapis toimub järelduste tegemine.

Kuna käesolevas töös teostatakse juhtumiuuring, kus juhtumiuuringu objektiks on mitmed üksikjuhtumid, siis see tähendab juhtumiuuringu läbiviimise kordamist üksikjuhtumite näol. Siinjuures tasub välja tuua, et juhtumite valimine ei toimu suvaliselt vaid teadlikult, kuna soovitakse, et uuritav juhtum oleks mõnes mõttes kas tüüpiline, kriitiline, ilmutav või unikaalne [19, lk. 40–42]. Sellele vaatamata valitakse tarkvaraarenduses paljud juhtumid siiski lihtsalt olemasolu või kättesaadavuse alusel [20]. Nõnda ka antud töö juures. Olenemata sellest, et töös lahendatav probleem on potentsiaalselt aktuaalne kõikides tarkvarajuurutus etappides, kus kasutatakse deklaratiivset paigaldust ja GitOpsi, pole juhtumite lihtne ei ole.

3.3 Juhtumi disain

Juhtumiuuringu üldine eesmärk on saada informatsiooni selle kohta, mida uuringu käigus loodetakse saavutada. See on defineeritud küsimuse või küsimustena ning sellele leitakse vastust kas andmete kogumise või analüüsi käigus.

Juhtumiuuringut alustatakse juhtumi disainiga. On defineerinud konkreetsed juhtumiuuringu disainielemendid (Lisa 2), millega tuleks juhtumiuuringu disaini faasis arvestada [14, ptk. 3]. Neist juhindutakse ka käesolevas töös ja nende kasutamist kajastatakse järgnevates alampeatükkides.

3.3.1 Eesmärk

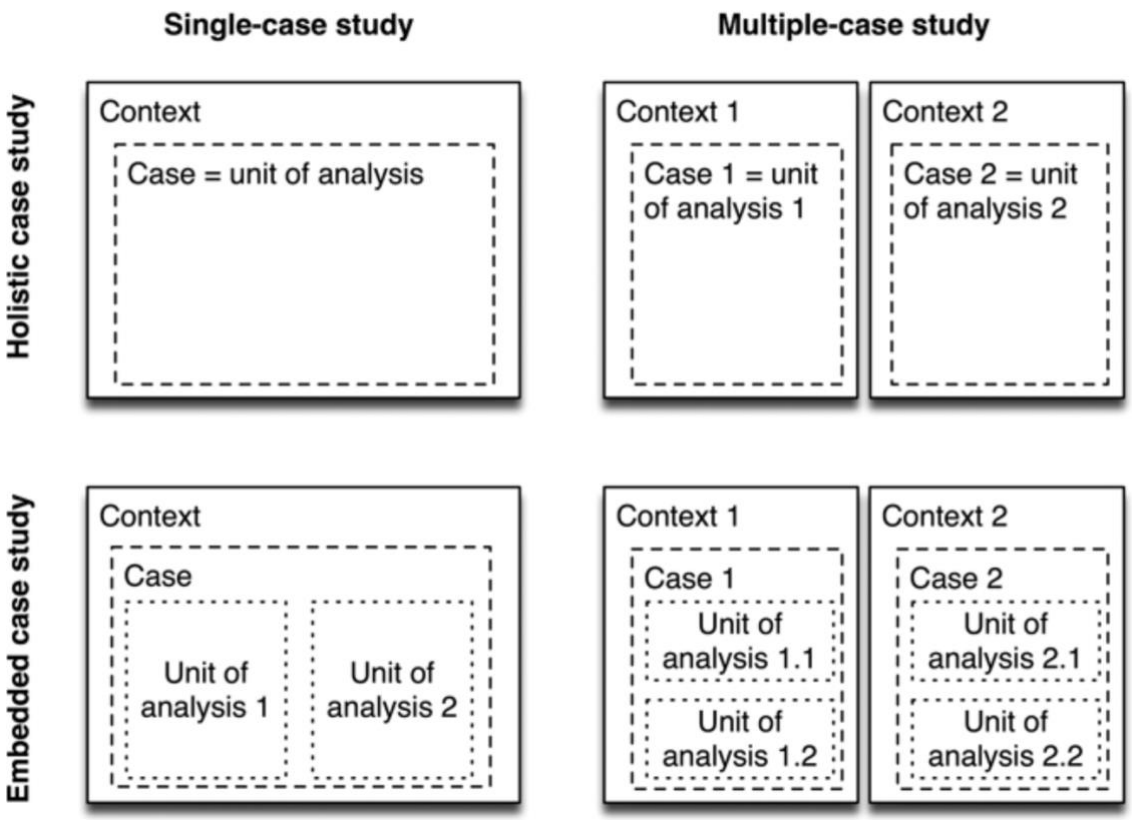
Juhtumiuuringu objektiks on mitmed üksikjuhtumid, kus üksikjuhtumiks on diplomtöös käsitletava probleemiga kokku puutunud Entigo OÜ projekt. Juhtumiuuringute läbiviimise põhjus on leida lahendus lõputöös sätestatud probleemile ehk soov on parandada ja tõhustada olemasolevate ja tulevaste projektide juurutusetappe. Juhtumiuuringud probleemi üksi küll ei lahenda, aga nad aitavad defineerida millistele nõuetele vastav lahend vastata võiks.

3.3.2 Analüüsiüksused

Juhtumiuuringu kontekstis tähendab analüüsiüksus konkreetse juhtumi konkreetset analüüsi. Eristatakse juhtumit ja analüüsiüksust. Üks juhtum võib sisaldada endas mitut analüüsiüksust.

Tuginedes Hösti ja teiste kirjeldatud näitele, kus tuuakse välja, et kui juhtumiuuringu kontekstiks on konkreetne ettevõte või rakenduse domeen [14, ptk. 3.2.3] ja sellele, et diplomitöös käsitletavad üksikjuhtumid pärinevad ühest ettevõttest saab järeldada, et kõik töös käsitletavad juhtumid on terviklikud. See tähendab seda, et iga juhtumi kohta on üks analüüsiüksus. Käesolevas töös on analüüsiüksusteks tarkvarajuurutusprotsessid.

Eristatakse veel terviklikke juhtumiuuringuid ja manustatud juhtumiuuringuid. [14, ptk. 3.2.3]. Joonisel 1 on terviklik juhtumiuuring (ülemine rida), manustatud juhtumiuuring (alumine rida), juhtumiuuring (vasak veerg) ja juhtumiuuring mitme üksikjuhtumiga (parem veerg).



Joonis 1. Juhtumiuuring ja juhtumiuuring mitme üksikjuhtumiga [19]

Käesolevas diplomitöös tehtav juhtumiuuring paigutub ülemisele reale parempoolsesse veergu, sest tegemist on juhtumiuuringuga, kus juhtumiuuringu objektiks on mitmed üksikjuhtumid. Seega iga üksikjuhtum sisaldab ühte analüüsiüksust.

3.3.3 Juhtumiuuringu küsimused

Projektide CI/CD ahelast on puudu deklaratiivset tarkvarajuurutus abistav ühtne utiliit, mis aitaks kirjeldada rakenduste soovitud olekud. Vastava tööriista ehitamiseks on vaja uurida probleemiga kokkupuutuvaid projekte. Selle lahendamiseks teostatakse juhtumiuuringud.

Juhtumiuuringu küsimus või küsimused annavad informatsiooni selle kohta, mida soovitakse uuringu käigus avastada. Nende teadmiste avastamine või omandamine näitab, et juhtumiuuring on saavutanud kavandatud eesmärgi [14, ptk. 3.2.5].

Käesoleva töö juhtumiuuringu küsimused on järgnevad:

- Küsimus 1: Kuidas saavatutakse juurutusprotsessieelsed rakenduste soovitud olekud praegu?

- Küsimus 2: Kuidas võiks rakenduste soovitud olekuid saavutada ideaalis?

Autori hinnangul on juhtumiuuringu küsimused hea viis kuidas tuua esile see, mida soovitakse avastada, aga on teadlasi, kes viivad on juhtumiuuringuid läbi ilma nendeta. Näiteks Shanks [21] ei kasuta oma juhtumiuuringutes uurimisküsimusi, vaid juhindub deduktsioonist ehk teooria kirjeldamisest liigutakse edasi ettepanekuteni, siis hüpoteesideni ning lõpuks jõutakse välja nende testimiseni.

3.3.4 Ettepanekud

Ettepanekud on küsimuste täpsemad vasted, pakkudes uurimusele täiendavaid detaile ja üksikasju [19]. Ettepanekud on ennustused või oletused, mis on juhtumiuuringu teooriast või taustast loogiliselt tuletatud [22]. Soovitatakse, et ettepanekud oleksid juhtumiuuringu küsimustega loogiliselt grupeeritud, kuna siis on lihtsam aru saada kust need on tuletatud [22]. Antud töös kasutatakse juhtumiuuringu küsimusi ja ettepanekuid.

Küsimused koos ettepanekutega:

- Küsimus 1: Kuidas saavutatakse juurutusprotsessi eelsed rakenduste soovitud olekud praegu?
    - Ettepanek 1.1: Kui rakenduste soovitud olekute saavutamiseks kasutatakse deklaratiivseid vahendeid, siis rakendatakse GitOpsi.
    - Ettepanek 1.2: Kui kasutatakse GitOpsi, siis rakenduste soovitud olekute kirjeldamiseks ei ole ühtset tööriista.
- Küsimus 2: Kuidas võiks rakenduste soovitud olekuid saavutada ideaalis?
    - Ettepanek 2.1: Rakenduste soovitud olekuid võiks saavutada ühtse käsurea utiliidi abil.
    - Ettepanek 2.2: Rakenduste soovitud olekuid soovitakse deklareerida kasutades deklaratiivseid vahendeid ja GitOpsi.

3.3.5 Andmete kogumise ja analüüsimise meetodid

Andmete kogumis meetodeid on kategoriseeritud kolme kategooriasse: otsene meetod (näiteks intervjuud), kaudne meetod (näiteks tööriista kasutamine), iseseisev meetod,

näiteks dokumentatsiooni analüüsimine [23]. Käesolevas töös kasutatakse andmekogumis meetoditena otsest meetodit ja iseseisvat meetodit ehk teostatakse intervjuud ja analüüsitakse koodi.

Intervjuud kui andmete kogumise meetod valiti sellpärast, et enamus uurimise jaoks vajalikke andmeid ei ole kuhugi üles kirjeldatud ja intervjuud on ainus variant soovitud teadmiste saamiseks.

Koodi analüüs valiti teiseks andmete kogumise meetodiks, kuna see aitab paremini aru saada millistele nõuetele diplomitöö tulemusena valmiv tööriist vastama peaks.

Juhtumiuuringu andmeid saab analüüsida erinevalt, aga käesoleva töö analüüsimeetodiks valiti *Cross-case synthesis* [19] ehk meetod, kus võrreldakse juhtumiuuringu objektiks olevaid üksikjuhtumeid. Valitud meetodi kasuks otsustati, sest töö autori hinnangul aitab see kõige paremini mõista ja üldistada üksikjuhtumite analüüsidelt saadud järeldusi.

3.3.6 Juhtumite ja andmete valimise strateegia

Nagu peatükis 3.2 Juhtumiuuringu läbiviimise protsess mainiti, siis juhtumite valimine peaks toimuma sihilikult, kui alati ei pruugi see võimalik olla ja seetõttu lähtutakse juhtumite saadavusest. Käesoleva töö juures valiti kolm projekti, mis autori hinnangul on piisav saamaks sisend lõputöö raames arendatava tööriista nõuete välja selgitamiseks.

Juhtumiuuringu intervjuude käigus intervjueeritakse konkreetse projekti kõige otsesemate inimestega kes osalesid juurutusetappi disainimisel ja ehitamisel. Lisaks intervjuudele uuritakse juhtumis käsitletava juurutusetapiga seotud koodi.

Juhtumite valiku olulisus tuleb välja siis, kui algab juhtumite kordamine. Juhtumeid saab korrata kas otseselt või teoreetiliselt. Otsene kordamine tähendab seda, et korduse läbiviimisel oodatakse sarnaseid tulemusi võrreldes esialgse juhtumiga. Teoreetiline kordamine tähendab seda, et oodatakse erisuguseid tulemusi. [19]

Käesolevas töös on kordamisstrateegiaks valitud otsene kordamine ehk oodatakse, et korratud juhtumid viitavad võrreldes esialgse juhtumiga sarnastele tulemustele.

Juhtumiuuringu üksikjuhtumite intervjuudest ja koodianalüüsist tehakse kokkuvõtted. Kvaliteedi ja usalduse tagamiseks palutakse intervjueeritavatel hiljem vastavad

kokkuvõtted üle vaadata. Vajadusel saavad nad lisada neile omapoolsed kommentaarid või märkused.

3.4 Üksikjuhtumid

Käesolevad peatükis kirjeldatakse kuidas koguti üksikjuhtumite andmeid ja milliseid järeldusi neist tehti.

3.4.1 Andmete kogumine

Nagu peatükis 3.3.5 välja toodi, siis kasutatakse andmete kogumise meetodina intervjuusid ja koodi analüüsimist. Intervjuud viiakse läbi avatud küsimustega ja poolstruktureeritud kujul. Avatud küsimusi kasutatakse sellepärast, et soovitakse, et intervjueeritavad ei oleks oma vastustega kuidagi piiritletud. Poolstruktureeritud kuju kasutatakse sellepärast, et see annab intervjuule piisavalt vabadust, kuid kindlustab selle, et intervjuul oleksid konkreetsed küsimused, millele soovitakse vastuseid. Kõik üksikjuhtumite intervjuud viidi läbi võttes arvesse Hösti ja teiste kirjeldatud intervjuude läbiviimissoovitusi [14, ptk. 4.3].

Juhtumiuuringut teostades viiakse läbi kolm üksikjuhtumit, kus igaühe käigus koguti toorandmeid. Tulenevalt töös kasutatavatest andmekogumismeetoditest kasutati vastavate andmete hankimiseks intervjuusid ja juurutusetapiga seotud koodi. Intervjuude läbiviimise jaoks koostas autor lihtsa meelespea (Lisa 3), mis abistaks teda nende läbiviimisel.

Juhtumiuuringute toorandmete organiseerimisel ja analüüsimisel teostatakse peaaegu alati nende puhastamine. Samuti ei pruugita eetilistel, loogilistel, privaatsusest tingitud või muudel põhjustel toorandmeid avalikustada. [14, ptk. 6.4]

Toorandmeid ei avalikustata ka käesoleva töö raames, kuid üksikjuhtumite toorandmete analüüsi käigus tehakse neist kokkuvõtted või aruanded, mis peegeldavad otseselt toorandmete sisu. Vastavad kokkuvõtted valideeritakse intervjueeritavate poolt, et tagada neile kvaliteet ja usaldus.

3.4.2 Andmete analüüsimine

Pärast toorandmetest puhastatud aruannete koostamist liigub fookus edasi andmete analüüsile.

Kuna kvalitatiivne analüüsimine sõltub erinevatest aspektidest, siis ei ole olemas lihtsat kirjeldust selle läbiviimiseks. Küll aga on mõned üldised tegevused, mis on asjakohased kõikidele analüüsitehnikatele. Näiteks üldistuste tegemine otsides mustreid ja seoseid, kasutades kategoriseerimist, kodeerimist ja kokkuvõtmist. Samuti aitab analüüsi mitmekordne läbiviimine ja andmete dokumenteerimine. [14, ptk. 5.3.1]

Eelnevas lõigus kirjeldatud tegevustest juhindutakse ka käesoleva töö üksikjuhtumite analüüsimisel. Selleks tehakse iga teostatud juhtumiuuringu analüüsi kohta vastavasisulised kokkuvõtted.

3.4.3 Juhtumiuuringute aruanded

Juhtumiuuringute aruanneteks on analüüsi käigus tehtud kokkuvõtted. Aruannete koostamiseks kirjeldas autor lihtsa juhendi (Lisa 4) millest kokkuvõtete koostamisel lähtuda. Aruanded on väljatoodud lisades (Lisa 5, Lisa 6, Lisa 7). Koostatud üksikjuhtumite aruannete alusel koostatakse juhtumiuuringu järeldused ja kokkuvõte.

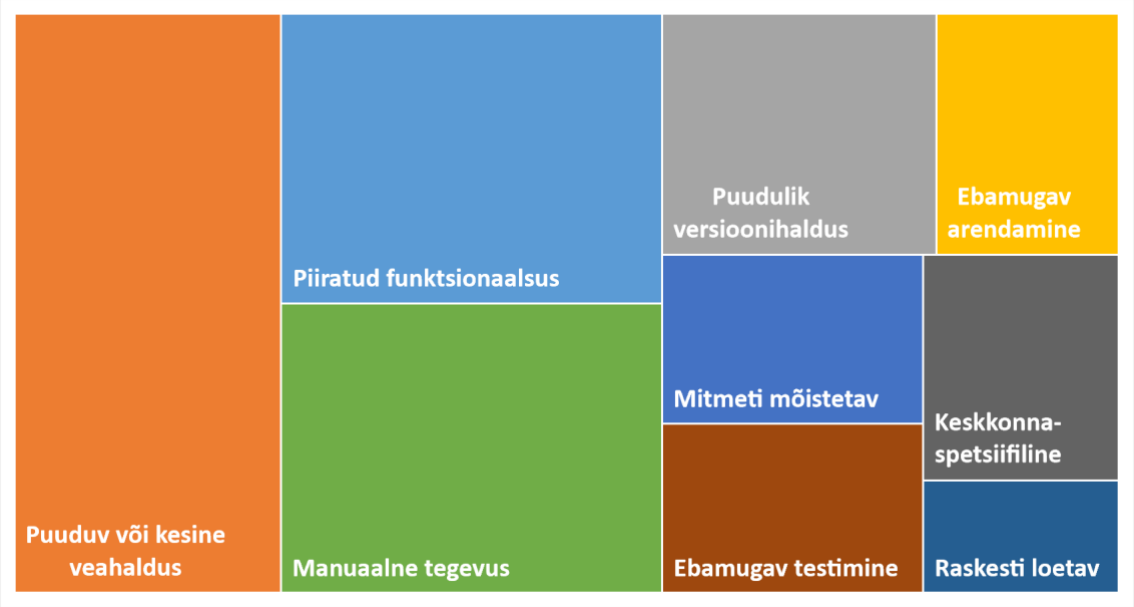
3.5 Juhtumiuuringu kokkuvõte

Juhtumiuuringu kokkuvõtmiseks ehk üksikjuhtumite põhjal järelduste tegemiseks on erinevaid viise. Antud töös lähtutakse *Cross-case synthesis*-st ehk tehnikast mille käigus teostatakse üksikjuhtumite võrdlemine. Juhtumeid ei esitleta eraldi vaid kokkuvõte koostamisel lähtutakse juhtumiuuringu küsimustest. Igale küsimusele vastamiseks arvestatakse kõikide üksikjuhtumitega [19].

Juhtumiuuringu läbiviimisel uuriti kolme üksikjuhtumit, kus üksikjuhtumiteks olid vaatluse all Entigo OÜ poolt arendatud projektide juurutusprotsessid. Uurimise läbiviimiseks teostati intervjuud ja olemasoleva lahenduse koodianalüüsid. Pärast seda teostati andmete puhastamine ja koondamine iseloomustavate märksõnadega kõrvutatud sektsioonidesse. Vastav tehnika aitab erinevate juhtumite üleselt kogutud andmeid paremini üldistada. Lisaks sellele jagati vastavad sektsioonid kolme gruppi: andmed ja märksõnad esialgsest lahendusest, andmed ja märksõnad analüüsitud koodist ning andmed ja märksõnad tööriista soovitud omadustest.

Juhtumiuuringu läbiviimisel kasutati küsimusi ja ettepanekuid. Esimene küsimus oli kuidas saavutatakse juurutusprotsessi eelsed rakenduste soovitud olekud? Läbiviidud

juhtumiuuringute üleselt võib väita, et soovitud olekute saavutamine oli vastavalt vajadusele kohandatud protsess, kus soovitud tulemuste ellu viimiseks kasutati erinevaid käsurea tööriistu, näiteks *sed*, *awk*, *grep*, *git* või *kubectl*. Uuritud lahendusi iseloomustavate märksõnade visualiseerimiseks koostati vastavasisuline puukaart (Joonis 2). Selle moodustamiseks kasutati üksikjuhtumite aruannetest, esialgse lahenduse ja analüüsitud koodi tabelitest (Lisa 5, Lisa 6, Lisa 7) saadud märksõnade esinemiste sagedusi. Uuritud lahendused küll rahuldasid vajadusi ja olid kohati taaskasutatavad, aga neil olid arvestatavad puudujäägid. Peamisteks vajakajäämisteks olid puuduv või kesine veahaldus, piiratud funktsionaalsus ja manuaalsed tegevused. See tähendas näiteks seda, et lahenduse edukuse või mitteedukuse veendumiseks tuli sageli teostada manuaalseid kontrolle. Vähem sagedasti esinevate märksõnade seas olid puudulik versioonihaldus, mitmeti mõistetavus ja keskkonna spetsiifiline. Versioonihalduse koha pealt ei tähendatud seda, et see oleks puudunud, vaid leiti, et lahenduste uute versioonide kasutamisel ei teadnud vana ja uus versioon üksteise olemasolust midagi. Mitmeti mõistetavus tähendas seda, et lahenduse koodi süntaksist tulenevate iseärasuste tõttu võis tekkida arusaamatusi ja soovimatuid tulemusi. Märksõna keskkonnaspetsiifiline tähendas seda, et lahendus oli kasutatav ainult konkreetses juurutuskeskkonnas.



Joonis 2. Üksikjuhtumites uuritud lahendusi iseloomustavate märksõnade puukaart

Esimese küsimusega seotud ettepanekud, ehk juhtumiuuringu kontekstis kui ennustused või oletused, pidasid mõlemad paika – rakenduste soovitud olekute saavutamiseks kasutati GitOpsi ja rakenduste soovitud olekute kirjeldamiseks puudus ühtne tööriist.

Puudus utiliit või töövahend, mis oleks olnud suuteline asendama üksikjuhtumites käsitletud rakenduste olekute deklareerimisega tegelenud lahendused.

Juhtumiuuringu teine küsimus oli kuidas võiks rakenduste soovitud olekuid saavutada ideaalis? Selle iseloomustamiseks kasutati märksõnu nagu taaskasutatav, vabavaraline, ühtne, stabiilne ja keskkonnast sõltumatu, kuid kõige sagedamini kasutatud märksõna oli soovitud funktsionaalsus. Üldistatult tähendab see seda, et sooviti töötavat ja stabiilset tööriista, mis oleks võimeline asendama olemasolevad lahendused, ja pakkuks midagi uut.

Lisaks viimati mainitud küsimusele olid sellega seotud veel kaks ettepanekut. Kuna sooviti, et erinevate projektide üleselt oleks üks ühtne tööriist, mis oleks kõigis samamoodi kasutatav, siis saab väita, et esimene ettepanek, kui rakenduste soovitud olekute saavutamiseks kasutatakse deklaratiivseid vahendeid, siis rakendatakse GitOpsi, oli tõene. Samuti oli tõene teine ettepanek, et kui kasutatakse GitOpsi, siis rakenduste soovitud olekute kirjeldamiseks ei ole ühtset tööriista. See tähendab seda, et töö kirjutamise ajal ei olnud töö autorile ega juhtumiuuringu käigus läbi viidud intervjueeritavatele teadaolevalt olemas lahendust, mis aitaks GitOps praktikaid kasutades lihtsa vaevaga deklareerida rakenduste soovitud olekuid. Sellise tööriista funktsionaalsetele ja mittefunktsionaalsetele nõuetele leitakse järgmises peatükis vastused.

4 Lahenduse analüüs

Lahenduse analüüsi osas tuginetakse juhtumite analüüsi käigus kogutud andmetele ja järeldustele. Nende abil kirjeldatakse nõuded lõputöö tulemusena valmivale tööriistale. Nõuete koostamise eesmärk on kirjeldada millistele funktsionaalsetele ja mittefunktsionaalsetele nõuetele diplomitöö lahendusena valmiv tööriist vastama peab. Kirjeldatud nõudeid kasutatakse tehnilise lahenduse peatüki sisendina. Nõuete koostamisel võeti aluseks Wiegers ja Beatty [24] kirjeldatud tarkvara nõuete spetsifikatsiooni mall (Lisa 8), mida kohandati vastavalt lõputöö mahule ja skoobile.

4.1 Üldine kirjeldus

Töö lahendusena valmib tööriist, mis on tarkvara juurutusprotsessides kasutatav GitOps praktikaid järgiv käsurea utiliit. Nimetatud haldusevahend on ennekõike mõeldud töötama CI/CD ahela automaatsetes protsessides, kuid vastavalt vajadusele saab seda kasutada ka manuaalsetes toimingutes. Tööriista peamine eesmärk on sisendargumentide alusel muuta konfiguratsioonifaile. Pärast tööriista poolt läbi viidud muudatuste teostamist ja deklareerimist oskab GitOps praktikaid järgiv Argo CD soovitud muudatused ellu viia. Argo CD on ehitatud kui Kuberntetese kontroller, mis jälgib töötavaid rakendusi võrreldes Kubernetese klastri seisu Giti versioonihalduses deklareeritud seisuga [13].

4.1.1 Töökeskkond

Tööriist peaks olema keskkonnast sõltumatu. Kuna haldusvahendi peamine kasutaja on CI/CD ahel, siis peaks olema seda võimalik kasutada erinevate teenuspakkujate keskkondades, näiteks Jenkinsis, GitHub Actions või Bitbucket Pipelines.

4.1.2 Disaini ja rakendamise piirangud

Arendatav haldusvahend peaks olema vabavaraline ja mahult võimalikult kompaktne. Vabavaraline sellepärast, et arendatud töörist oleks vabalt ja lihtsasti kasutatav erinevate osapoolte poolt ning mahult kompaktne sellepärast, et CI/CD ahelates jooksvad tööd ei peaks suuremahulise tööriista allalaadimisega tegelema.

4.2 Süsteemi omadused

Süsteemi omaduste detailsemaks kirjeldamiseks kasutati funktsionaalseid nõudeid. Gilb [25] soovitab üksikute nõuete defineerimiseks kasutada tekstipõhist hierarhilist märgistamisskeemi. Nimetatud tehnikat kasutati tööriistaga seotud funktsionaalsete nõuete kirjeldamiseks. Tööriistal peab olema neli omadust ehk käsurea käsku: uuendamine, kopeerimine, kustutamine ja abi. Neist omadustest lähtuvalt kategoriseeriti tööriista funktsionaalsed nõuded.

Omadus – uuendamine

Uuendamise eesmärgiks on sisendargumentide alusel uuendada Kubernetese konfiguratsioonifailides asuvate kujutiste versioone. Nagu GitOpsile kohane, siis konfiguratsioonifailid asuvad Gitis ehk lihtsustatult näeb uuendamise protsess välja järgmine:

1. hangitakse versioonihaldusest olemasolevad konfiguratsiooni failid;
2. teostakse soovitud uuendused;
3. muudatused laetakse versioonihaldusesse.

Sellest protsessist lähtuvalt kirjeldati uuendamisega seotud funktsionaalsed nõuded (vt Lisa 9).

Omadus – kopeerimine

Kopeerimisel teostatakse olemasolevate Kubernetese konfiguratsioonide dubleerimine ja pärast seda nende modifitseerimine. Selle tegevuse mõte on uute keskkondade kiire ja veatu ülesseadmine. Kopeerimise protsess lihtsustatul on järgnev:

1. hangitakse versioonihaldusest olemasolevad konfiguratsiooni failid;
2. valitud konfiguratsioonidest tehakse koopia;
3. kopeeritud konfiguratsioone muudetakse vastavalt sisendile;
4. muudatused laetakse versioonihaldusesse.

Kopeerimisega seotud funktsionaalsed nõuded on kirjeldatud lisas 10.

Omadus – kustutamine

Kustutamise eesmärk on Kubernetese ja Argo CD konfiguratsioonide eemaldamine repositooriumist. Selle tegevuse mõte on sisendargumentide alusel valitud keskkonna eemaldamine. Kustutamisega seotud funktsionaalsed nõuded on kirjeldatud lisas 11.

Omadus – abistamine

Abistamise eesmärk on anda informatsiooni tööriista kasutamise kohta. Abistamise käsk loetleb üles kõik tööriista poolt kasutatavad käsud ja nende poolt toetatud sisendargumendid. Abistamisega seotud funktsionaalsed nõuded on kirjeldatud lisa 12.

4.3 Andmete nõuded

Tööriista andmeteks on Kubernetese objektide yaml-konfiguratsioonid ja tööriista sisendargumendid (Lisa 13).

Teistest käskudest mõnevõrra erilisem on kopeerimise käsk, mis kasutab paigaldamise konfiguratsiooni. See tähendab seda, et kopeerimise ühe sisendina kasutatakse tekstifaili, mis aitab pärast soovitud andmete kopeerimist modifitseerida selle sisu. Kuna paigaldamise konfiguratsioonid pärinevad varem kasutusel olnud lahendustest, siis esialgu võetakse kasutusele just need.

4.4 Väliste liideste nõuded

Siin peatükis kirjeldatakse millised on arendatava tööriista liidesed.

Kuna töö tulemusena valmiv lahendus on käsurea tööriist, siis sellest tulenevalt on selle kasutajaliides tekstipõhine. Tööriista kasutamiseks on vaja kirjutada tööriista poolt toetatud tekstil põhinevaid käske.

Tööriista hakatakse peamiselt kasutama Dockeri konteineris ehk operatsioonisüsteemi tasemel virtualiseeritud keskkonnas. Tööriist paigutatakse Dockeri kujutisse, mida hakatakse kasutama erinevates CI/CD ahelates. See tähendab seda, et standardolukorras oleks tööriista kasutamiseks vaja Dockeri tuge. Küll aega ei tähenda see seda, et tööriist peaks olema kasutatav ainult Dockeri kujutisena.

4.5 Kvaliteedi omadused

Käesolevas peatükis sõnastatakse tööriista mittefunktsionaalsed nõuded. Kirjeldatakse tööriista kasutatavuse, turvalisuse, ohutuse ja lokaliseerumisega seotud nõuded.

Kasutatavus

Tööriista kasutatavusega seotud omadused:

- Tööriist peaks olema vabavaraline ja kasutatav võimalikult paljudes keskkondades.
- Tööriist peaks andma teavet selle kohta millised muudatused teostati.
- Tööriist peaks olema mahult võimalikult väike ehk mõnikümmend megabaiti.

Turvalisus

Tööriist peaks töödeldavaid andmeid kasutama heaperemehelikult ehk kasutama neid ainult selleks ettenähtud otstarbel.

Ohutus

Tööriist ei tohi oma tööprotsesside käigus tekitada selle kasutajale võimaliku kahju.

Rahvusvahelisuse ja lokaliseerumise nõuded

Tööriista lähtekood ja kasutajaliides peab olema kirjutatud inglise keeles.

5 Tehniline teostus

Siin peatükis kirjeldatakse kuidas teostati tehnilise lahenduse planeerimine ja kuidas viidi ellu selle realiseerimine. Käesolev peatükk tugineb kõikidele eelnevatele peatükkidele, kuid eelkõige lahenduse analüüsile.

5.1 Tehnilise lahenduse planeerimine

Enne tehnilise lahenduse realiseerimist on vaja teostada tehniline planeerimine. See tähendab seda, et tuleb välja selgitada, millises programmeerimis keeles lahendus teostada ja milline peaks olema selle arhitektuur.

5.1.1 Programmeerimiskeel

Programmeerimiskeeleks valiti Golang ehk Go. Viimase kasuks otsustamisel aitas selle lihtsus ja kiirus [26], kuid kõige mõjuvaimaks argumendiks oli tema integreerimise potentsiaal Argo CD-ga. Kuna Argo CD on samuti Go-s kirjutatud, siis tutvudes selle lähtekoodiga [27] tundus esmasel uurimisel, et töö tulemusena valmiva rakenduse liidestamine Argo CD-ga on kõige mõistlikum kasutades just Golangi. Teine oluline argument, miks Golang tundus sobiv valik, oli nagu ennist juba mainitud tema lihtsus. Lahenduse analüüsist selgus, et töö tulemusena valmiv tööriist peaks olema vabavaraline. Autori hinnangul on hea vabavaraline tarkvara lihtsasti loetav, selgesõnaline ja üheselt mõistetav. Go disainimisel on sarnaste printsiipidega arvestatud.

Go disainimisel arvestati, et tarkvaraprojektid on suured ja paljude arendajatega. Selle tõttu loodi keel võimalikult lihtne ja minimalistlik. See julgustab arendajaid lihtsusele ja tootlikkusele, vältides sellega segadust ja keerukust [26]. Seega Go ei sisalda, või sisaldab vähesel määral, oma süntaksis keerulisi ja süvenemist vajavaid tehnilisi konstruktsioone. See loob eelduse, et tööriista lähtekood on lihtsasti loetava, arusaadav ja keeleliselt tuttav.

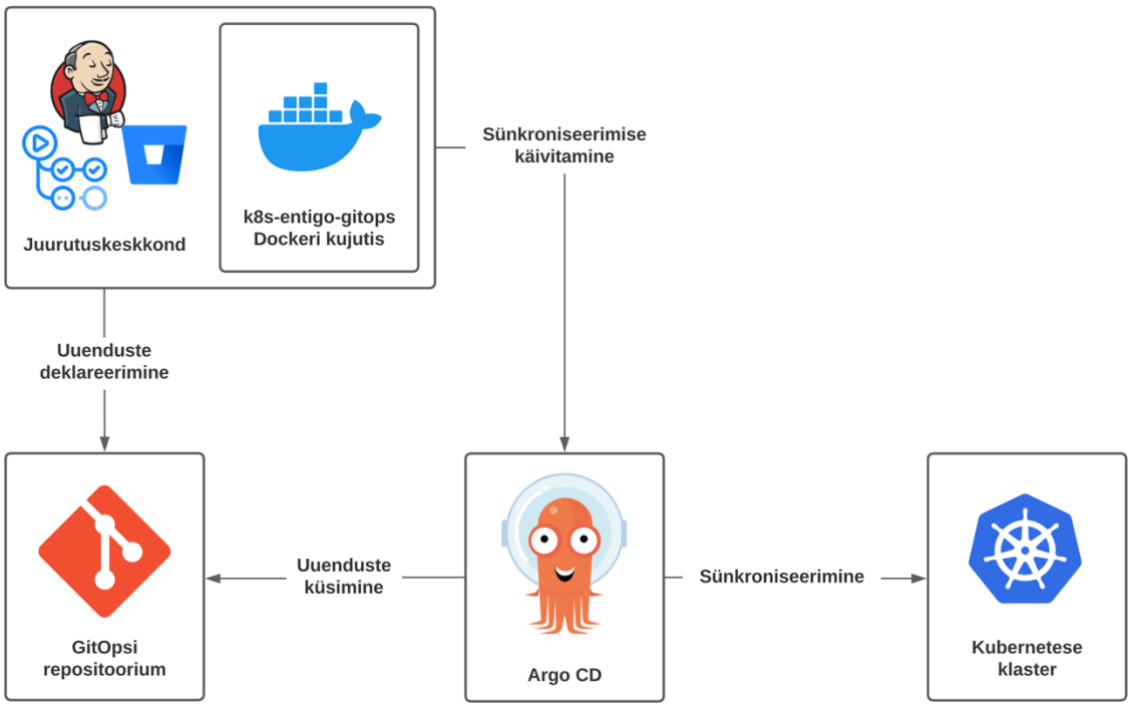
5.1.2 Virtualiseerimine

Paljud enam levinud CI/CD keskkonnad toetavad enda tööprotsessides Dockeri kujutisi [28], [29], [30]. Seetõttu tundus mõistlik paigutada lahendusena valmiv tööriist samuti Dockeri kujutisse, kuna see annab CI/CD ahela töödele võimekuse teostada tööriista poolt tehtud toimingud isoleeritud Dockeri konteineris.

5.1.3 Lahenduse arhitektuur

Lahendus arhitektuur koosneb viiest komponendist (Joonis 3):

- Juurutuskeskkond ehk CI/CD ahel.
- Lahendusena valmiv tööriist, mille nimi on k8s-entigo-gitops. See laetakse alla Dockeri kujutisena, mida juurutuskeskkond kasutab ühe oma sammuna.
- GitOpsi repositooriumist, kuhu deklareeritakse rakenduste soovitud olekud.
- Argo CD, mis hoolitseb selle eest, et GitOpsi repositooriumis kirjeldatud olek jõuaks Kubernetese klastrisse.
- Kubernetese klaster, kus asuvad erinevad keskkonnad ja rakendused.



Joonis 3. Lahenduse arhitektuur

Jooniselt 3 on näha, et Argo CD saab kahel viisil aru, et tuleks alustada sünkroniseerimisega. Argo CD on võimalik seadistada selliselt, et ta käiks kas ise kontrollimas, et GitOpsi repositoorium on võrreldes klastri versiooniga sünkroonist väljas, või siis käivitatakse sünkroniseerimine CI/CD ahela poolt automaatselt.

5.2 Lahenduse realiseerimine

Pärast seda kui kõik lahenduse planeerimisega seotud tegevused olid teostatud võis alata lahenduse realiseerimine.

5.2.1 Tööriista sõltuvused

Enne lahenduse arendamist uuriti kui palju tööriista loogikast on võimalik ja mõistlik erinevate väliste sõltuvuste abil ära katta. Sõltuvused on tööriista toimimiseks vajaminevad välised tarkvarakomponendid. Sõltuvusteks võivad olla näiteks tarkvarateegid või tarkvarapaketid.

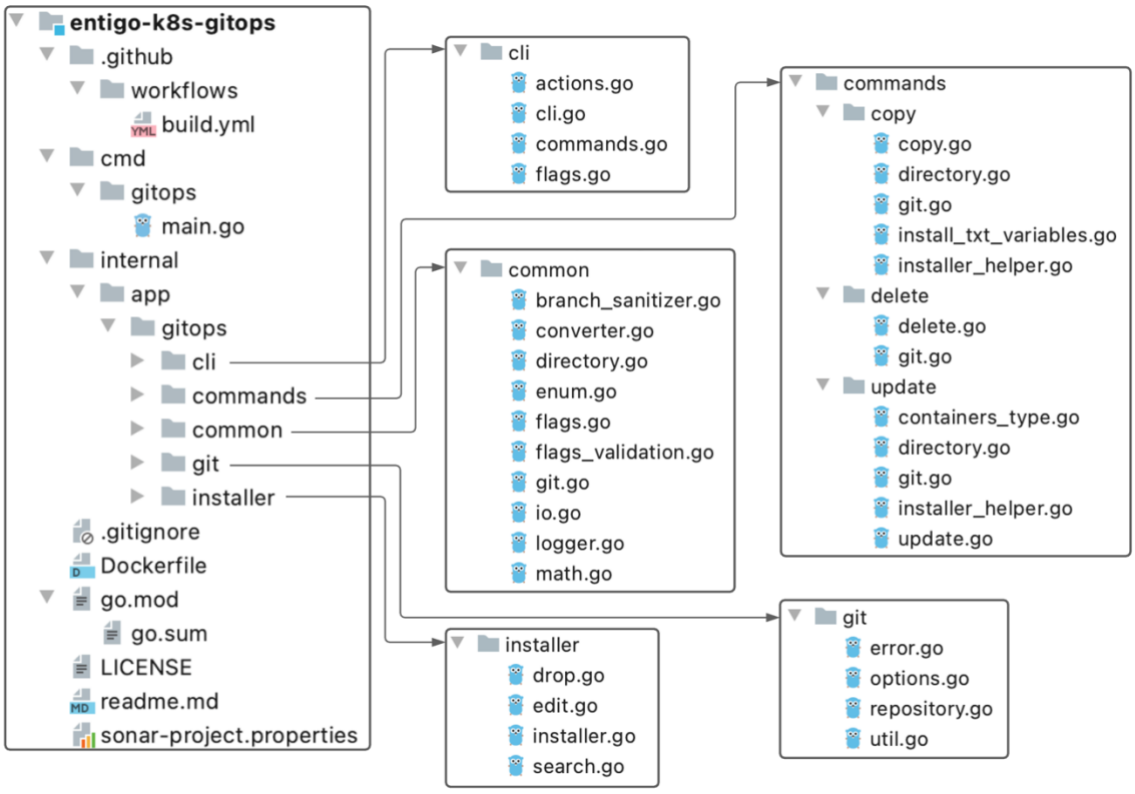
Võimalike sõltuvuste otsimiseks kasutati erinevaid allikaid, aga üks kasulikumaid oli Awesome Go [31], mis on kureeritud loetelu erinevatest Go raamistikest, teekidest ja tarkvaradest. Lahenduse väliste sõltuvuste kasutamisel lähtuti eesmärgist kasutada neid võimalikult vähe, aga nii palju kui vaja. Analüüsi käigus välja selgitatud nõuetest lähtuvalt kasutati tööriista juures viit otsest sõltuvust:

- github.com/go-git/go-git [32] – sõltuvust kasutati Giti operatsioonide teostamiseks
- github.com/otiai10/copy [33] – sõltuvust kasutati kaustade kopeerimiseks kopeerimiseks
- github.com/urfave/cli [34] – sõltuvust kasutati käsurea tööriista funktsionaalsuste lihtsamaks ja kiiremaks ehitamiseks
- golang.org/x/crypto [35] – sõltuvust kasutati *StrictHostKeyChecking* sisse ja välja lülitamiseks
- github.com/go-yaml/yaml/tree/v3 [36] – sõltuvust kasutati YAML-failide töötlemiseks

Siinjuures tasub välja tuua, et kaudsete sõltuvuste hulk, ehk sõltuvuste mida kasutavad otsesed sõltuvuse, on oluliselt suurem.

5.2.2 Struktuur

Projekti struktuuri loomisel võeti eeskuju *golang-standards/project-layout* [37] kirjeldatud lähenemisest. Kuna Go ei luba tsüklilisi sõltuvusi, siis tuli tööriista arendamisel ja koodi organiseerimisel sellest teadmisest lähtuda. Joonisel 4 on välja toodud tööriista projekti struktuur. Github faililaiendiga kaust sisaldab tööriista avaldamisega seotud koodi, *Dockerfile* sisaldab endas Dockeri kujutise loomisega seotud koodi, *sonar-project.properites* sisaldab endas koodi kvaliteedi kontrollimisega seotud konfiguratsiooni. *License* on tööriista litsents ja *readme.md* on tööriista ja selle kasutamise kohta teavet andev tekstidokument. Ülejäänud kaustad ehk paketid on Go spetsiifilised ja need sisaldavad endas tööriista koodi. *Cmd* kaust sisaldab endas tööriista käivitamiseks vajalikku koodi. *Cli* nimelises kaustas on käsurea funktsionaalsusega seotud kood ja *commands* kaustas on tööriista käskudega seotud kood. *Common* kaustas asub tööriista kõikide pakkete poolt kasutatav kood. Giti kaustas asub giti operatsioonidega seotud kood ja *installer* kaustas on failimuudatuste tegemiseks vajaminev kood.



Joonis 4. Tööriista projekti struktuur

5.2.3 Arendus

Tööriista arendamist alustati käsureafunktsionaalsusest ehk tekstipõhisest kasutajaliidesest. Selle jaoks arendati loogika, mis oskas kasutaja sisendi peale vastavalt reageerida. Algselt läheneti sellele kasutades Go standardis olevaid teeke, aga kuna sisendargumentide ja abistava käsu loogika hulk kasvas, siis lõpuks võeti kasutusele ikkagi väline sõltuvus - *urfave/cli*. Viimase abiga oli suur osa tekstipõhisest kasutajaliidesest kerge vaevaga teostatav. Pärast seda tekkis vajadus muudetavate konfiguratsioonide allalaadimiseks. Kuna failid asusid Giti repositooriumis, siis tuli realiseerida Giti toimingutega seotud loogika. Kui failid olid hangitud, siis sai alustada tööriista käskudest tuleneva loogika arendamisega. See tähendas seda, et konfiguratsiooni faile tuli vastavalt vajadusele muutma hakata. Selle ülesande lahendamisele prooviti läheneda mitut moodi, kuid üsna pea saadi aru, et lihtsat otseteed sellele ei ole. Kasutusele tuli võtta *go-yaml/yaml* nimeline väline sõltuvus, mis aitas YAML konfiguratsioonifailid viia Go's kirjeldatud puustruktuurile (Joonis 5). See tegi nende muutmise ja töötlemise kiiremaks ja mugavamaks, kui teiste katsetatud lähenemistega. Viimase abiga realiseeriti uuendamise ja kopeerimisega seotud loogika. Pärast soovitud failimuudatustega seotud loogika arendamist oli suurem ja keerulisem osa tööst teostatud. Pärast seda järgnesid veel mõned parandused, muudatused ja täiendused.

6 Tulemused

Käesolev peatükk hindab kas töös kasutatud uurimismeetod täitis oma eesmärgi ehk andis piisava sisendi lahenduse analüüsi peatükis kirjeldatud nõuete defineerimiseks ning kas kirjeldatud nõuded olid piisavaks sisendiks arendustegevusega alustamiseks. Viimaseks hinnatakse kas diplomitöö tulemusena valminud tehniline lahendus vastas nõuetele ja aitas lahendada töös kirjeldatud probleemi.

6.1.1 Uurimismeetodi hindamine

Käesoleva töö uurimismeetodiks kasutati juhtumiuuringut, kus juhtumiuuringu objektiks on mitmed üksikjuhtumid. Selle eesmärk oli välja selgitada milline oli rakenduste soovitud oleku deklareerimise protsess enne ja milline võiks see olla ideaalis. Kogutud andmed ja järeldused olid piisavad, et nende põhjal teostada lahenduse analüüs ja kirjeldada tööriista nõuded. Üksikjuhtumite andmete võrdlemine näitas, et andmed on üksteisega võrreldes üsna sarnased. See andis ühelt poolt kindlust, et töös kirjeldatud probleem on reaalne, ja teisalt, et tööriistale soovitud nõuded on võrdlemisi sarnased.

Lisaks võib välja tuua, et ideaalis oleks võib olla veel mõne üksikjuhtumi kasutamine olnud hea variant, kuna see oleks potentsiaalselt veelgi enam aidanud täpsustada ja kinnitada tööriista nõudeid. Kuna töö skoop oli aga piiritletud, siis täiendavate üksikjuhtumite läbiviimist ei peetud mõistlikuks.

6.1.2 Tehnilise lahenduse hindamine

Tööriista hindamisel võeti aluseks lahenduse analüüsi peatükis kirjeldatud nõuded ja see kas see vastab töös kirjeldatud probleemile. Lahenduse analüüsi nõuetest järeldades sooviti, et arendatud tööriistal oleks kaks lihtsamat ja kaks keerulisemat käsku ehk siis vastavalt abi ja kustutamine ning uuendamine ja kopeerimine. Kõik lahenduse analüüsi käigus püstitatud nõuded said täidetud ja tegelikult rohkemgi veel. Töö tulemusena valmis vabavaraline tööriist, mille Dockeri kujutist [38] kasutatakse töös käsitletud üksikjuhtumite juurutuskeskkondades ja uutes projektides, mida töös ei kajastatud.

6.1.3 Tööriista laiendused ja edasised tegevused

Arendatud tööriist on juba kasutusel ja lahendab töös kirjeldatud probleemi, aga see ei tähenda, et tööriista edasiarendamine ei jätkuks. Käesoleva töö raames arendatud tööriist

tegeles peaasjalikult soovitud olekute deklareerimisega. Järgmiseks oleks vaja arendada Argo CD spetsiifiliste osa ehk funktsionaalsus, mis oskaks Argo CD poolt tehtavaid toimingud käivitada. Lisaks oleks vaja kirjutada *end-to-end* testid ja muuta *readme* dokument uutele kasutajatele lihtsamini arusaadavaks. Soovitusliku poole pealt võiks tööriista struktuuri veelgi lihtsustada. Töö kirjutamise ajal on lähtekood märgistatud *pre-release* sildiga. Kui siin alapeatükis kirjeldatud sammud saaksid tehtud, siis võiks kaaluda viimati mainitud sildi eemaldamist.

7 Kokkuvõte

Deklaratiivne tarkvarajuurutamine tähendab tarkvara soovitud oleku saavutamist deklaratiivsete vahendite abil. Töös käsitletud probleem tulenes reaalsest olukorrast, kus tarkvaraprojektide juurutusetappidest puudus soovitud olekute saavutamiseks ühtne ja stabiilne haldusvahend. Sellest tulenevalt oli töö peamine eesmärk kirjeldatud probleemi lahendav tööriist ehitada.

Probleemi paremaks mõistmiseks ja tööriista ehitamiseks vajalike nõuete kogumiseks oli tarvis seda lähemalt uurida. Selle ülesande lahendamise valiti juhtumiuuring, kus juhtumiuuringu objektiks on mitmed üksikjuhtumid. Üksikjuhtumite käigus viidi läbi probleemiga kokku puutunud projektide uurimine ehk andmete kogumine, mille käigus teostati intervjuud ja koodianalüüsid. Juhtumiuuringu kokkuvõtmiseks teostati üksikjuhtumite võrdlus, mis näitas, et töös kirjeldatud probleem on reaalne ja, et tehnilisele lahendusele soovitud ootused ja nõuded on võrdlemisi sarnased.

Juhtumiuuringule järgnes lahenduse analüüs, kus kogutud andmete põhjal kirjeldati töö tulemusena valmiva tööriista funktsionaalsed ja mittefunktsionaalsed nõuded.

Pärast nõuete koostamist algas tehnilise lahenduse planeerimine. Valiti tööriista arendamiseks sobilik programmeerimiskeel ja kirjeldati lahenduse kõrgetasemeline arhitektuur. Sellele järgnes lahenduse realiseerimine ehk kirjeldati millised olid tööriista arendamise juures kasutatud sõltuvused, milline oli projekti struktuur ning kuidas toimus tööriista arendamine.

Viimaseks toodi välja töö tulemused, kus leiti, et töös kasutatud uurimismeetod oli sobilik töö läbiviimiseks vajalike andmete kogumiseks ja probleemi analüüsimiseks. Saadud tulemused olid eelduseks tehnilise lahenduse nõuete defineerimiseks ja tööriista arendamiseks.

Töö tulemusena valmis avatud lähtekoodiga tööriist [39] ja seda sisaldavad Dockeri kujutised [38] mida kasutatakse nii probleemiga kokku puutunud projektides, kui uutes lahendustes.

Kasutatud kirjandus

[1] G. Kim, J. Humble, P. Debois, and J. Willis, *DevOps Handbook*. IT Revolution Press, 2016 [Võrgumaterjal]. Kättesaadav: https://learning.oreilly.com/library/view/the-devops-handbook/9781457191381/DOHB-0-FM-introduction.xhtml. [Kasutatud 10.04.2021]

[2] B. Laster, 'Continuous Delivery in Kubernetes with ArgoCD', *Continuous Delivery in Kubernetes with ArgoCD*. [Võrgumaterjal]. Kättesaadav: https://www.oreilly.com/attend/continuous-delivery-in-kubernetes-with-argocd/0636920511274/0636920052484/. [Kasutatud 21.04.2021]

[3] J. Davis and R. Daniels, *What Is DevOps?* O'Reilly Media, Inc., 2018 [Võrgumaterjal]. Kättesaadav: https://learning.oreilly.com/library/view/what-is-devops/9781492039891/ch01.html. [Kasutatud 22.04.2021]

[4] M. Hüttermann, *DevOps for Developers*. Apress, 2012 [Võrgumaterjal]. Kättesaadav: https://learning.oreilly.com/library/view/devops-for-developers/9781430245698/9781430245698_Ch01.xhtml. [Kasutatud 22.04.2021]

[5] 'Devopsdays Ghent 2009', *devopsdays*. [Võrgumaterjal]. Kättesaadav: https://legacy.devopsdays.org/events/2009-ghent/. [Kasutatud 22.04.2021]

[6] I. Buchanan, 'History of DevOps', *History of DevOps*. [Võrgumaterjal]. Kättesaadav: https://www.atlassian.com/devops/what-is-devops/history-of-devops. [Kasutatud 22.04.2021]

[7] C. Endres, U. Breitenbucher, M. Falkenthal, O. Kopp, F. Leymann, and J. Wettinger, 'Declarative vs. Imperative: Two Modeling Patterns for the Automated Deployment of Applications', p. 7.

[8] 'Guide To GitOps', *Weaveworks*. [Võrgumaterjal]. Kättesaadav: https://www.weave.works/technologies/gitops/. [Kasutatud 23.04.2021]

[9] N. Yellavula, *IaC with Terraform - Hands-On RESTful Web Services with Go*, Second Edition. Packt Publishing, 2020 [Võrgumaterjal]. Kättesaadav: https://learning.oreilly.com/library/view/hands-on-restful-web/9781838643577/cover.xhtml. [Kasutatud 23.04.2021]

[10] K. Morris, *Infrastructure as Code*, 2nd Edition. O'Reilly Media, Inc., 2020 [Võrgumaterjal]. Kättesaadav: https://learning.oreilly.com/library/view/infrastructure-as-code/9781098114664/titlepage01.html. [Kasutatud 23.04.2021]

[11] 'What is Kubernetes?', *Kubernetes*. [Võrgumaterjal]. Kättesaadav: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/. [Kasutatud 23.04.2021]

[12] 'GitOps', *GitOps*. [Võrgumaterjal]. Kättesaadav: https://www.gitops.tech/. [Kasutatud 23.04.2021]

[13] Argo CD, 'Overview'. [Võrgumaterjal]. Kättesaadav: https://argoproj.github.io/argo-cd/. [Kasutatud 23.04.2021]

[14] M. Höst, B. Regnell, P. Runeson, and A. Rainer, *Case Study Research in Software Engineering: Guidelines and Examples*. Wiley, 2012 [Võrgumaterjal].

Kättesaadav: https://learning.oreilly.com/library/view/case-study-research/9781118104354/. [Kasutatud 27.04.2021]
[15] S. McCombes, ‘How to Do a Case Study’, *Scribbr*, 08-May-2019. [Võrgumaterjal]. Kättesaadav: https://www.scribbr.com/methodology/case-study/. [Kasutatud 29.04.2021]
[16] W. Ellet, *The Case Study Handbook, Revised Edition*. Harvard Business Review Press, 2018 [Võrgumaterjal]. Kättesaadav: https://learning.oreilly.com/library/view/the-case-study/9781633696167/Text/chapter02.html. [Kasutatud 29.04.2021]
[17] G. Vega, *The Case Writing Workbook : A Self-Guided Workshop*, 2nd Edition. Routledge, 2017 [Võrgumaterjal]. Kättesaadav: https://www.taylorfrancis.com/books/mono/10.4324/9781315455891/case-writing-workbook-gina-vega. [Kasutatud 29.04.2021]
[18] S. Virkus, ‘Juhtumiuuringud’, 2010. [Võrgumaterjal]. Kättesaadav: https://www.tlu.ee/~sirvir/Infootsingu%20teooria/Infokaitumise,%20info%20hankimise%20ja%20%20otsingu%20ning%20infopadevuse%20uurimise%20meetodid/juhtumiuuringud.html. [Kasutatud 29.04.2021]
[19] R. K. Yin, *Case Study Research: Design and Methods*, 3rd edition., vol. 5. SAGE Publications, 2003.
[20] I. Benbasat, D. K. Goldstein, and M. Mead, ‘The Case Research Strategy in Studies of Information Systems’, *MIS Q.*, vol. 11, no. 3, pp. 369–386, 1987, doi: 10.2307/248684.
[21] G. Shanks, ‘Guidelines for Conducting Positivist Case Study Research in Information Systems’, *Australas. J. Inf. Syst. Vol 10 No 1 2002*, vol. 10, Nov. 2002, doi: 10.3127/ajis.v10i1.448.
[22] J. M. Verner, J. Sampson, V. Tosic, N. A. A. Bakar, and B. A. Kitchenham, ‘Guidelines for industrially-based multiple case studies in software engineering’, in *2009 Third International Conference on Research Challenges in Information Science*, 2009, pp. 313–324, doi: 10.1109/RCIS.2009.5089295.
[23] T. Lethbridge, S. Sim, and J. Singer, ‘Studying Software Engineers: Data Collection Techniques for Software Field Studies’, *Empir. Softw. Eng.*, vol. 10, pp. 311–341, Jul. 2005, doi: 10.1007/s10664-005-1290-x.
[24] K. E. Wiegers and J. Beatty, *Software Requirements*, 3rd Edition. Microsoft Press, 2013 [Võrgumaterjal]. Kättesaadav: https://learning.oreilly.com/library/view/software-requirements/9780735679658/. [Kasutatud 01.05.2021]
[25] T. Gilb, ‘Principles of Software Engineering Management’, 1988.
[26] M. A. Titmus, *Cloud Native Go*. O’Reilly Media, Inc., 2021 [Võrgumaterjal]. Kättesaadav: https://learning.oreilly.com/library/view/cloud-native-go/9781492076322/. [Kasutatud 07.05.2021]
[27] *argoproj/argo-cd*. Argo Project, 2021 [Võrgumaterjal]. Kättesaadav: https://github.com/argoproj/argo-cd. [Kasutatud 07.05.2021]
[28] ‘Use Docker images as build environments | Bitbucket Cloud’, *Atlassian Support*. [Võrgumaterjal]. Kättesaadav: https://support.atlassian.com/bitbucket-cloud/docs/use-docker-images-as-build-environments/. [Kasutatud 08.05.2021]
[29] ‘Using Docker with Pipeline’, *Using Docker with Pipeline*. [Võrgumaterjal]. Kättesaadav: https:/www.jenkins.io/doc/book/pipeline/docker/. [Kasutatud 08.05.2021]

[30] ‘Run your CI/CD jobs in Docker containers | GitLab’. [Võrgumaterjal]. Kättesaadav: https://docs.gitlab.com/ee/ci/docker/using_docker_images.html. [Kasutatud 08.05.2021]
[31] Avelino, *avelino/awesome-go*. 2021 [Võrgumaterjal]. Kättesaadav: https://github.com/avelino/awesome-go. [Kasutatud 08.05.2021]
[32] *go-git/go-git*. go-git, 2021 [Võrgumaterjal]. Kättesaadav: https://github.com/go-git/go-git. [Kasutatud 09.05.2021]
[33] H. OCHIAI, *otiai10/copy*. 2021 [Võrgumaterjal]. Kättesaadav: https://github.com/otiai10/copy. [Kasutatud 09.05.2021]
[34] *urfave/cli*. urfave, 2021 [Võrgumaterjal]. Kättesaadav: https://github.com/urfave/cli. [Kasutatud 09.05.2021]
[35] ‘crypto · pkg.go.dev’. [Võrgumaterjal]. Kättesaadav: https://pkg.go.dev/golang.org/x/crypto. [Kasutatud 09.05.2021]
[36] ‘yaml.v3 - gopkg.in/yaml.v3’. [Võrgumaterjal]. Kättesaadav: https://gopkg.in/yaml.v3. [Kasutatud 09.05.2021]
[37] *golang-standards/project-layout*. golang-standards, 2021 [Võrgumaterjal]. Kättesaadav: https://github.com/golang-standards/project-layout. [Kasutatud 08.05.2021]
[38] ‘entigolabs/entigo-k8s-gitops’. [Võrgumaterjal]. Kättesaadav: https://hub.docker.com/r/entigolabs/entigo-k8s-gitops. [Kasutatud 08.05.2021]
[39] ‘entigolabs/entigo-k8s-gitops’, *GitHub*. [Võrgumaterjal]. Kättesaadav: https://github.com/entigolabs/entigo-k8s-gitops. [Kasutatud 09.05.2021]
[40] ‘go-yaml/yaml’, *GitHub*. [Võrgumaterjal]. Kättesaadav: https://github.com/go-yaml/yaml/blob/v3/yaml.go. [Kasutatud 11.05.2021]

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Mina, Kert Kukk

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Deklaratiivset tarkvarajuurutamist abistava tööriista analüüs ja arendus“, mille juhendaja on Kristiina Hakk ja kaasjuhendaja on Rein Remmel.
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

17.05.2021

1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.

Lisa 2 – Juhtumiuuringu disainielemendid

Tabel 1. Juhtumiuuringu disainielementide tabel [14, ptk. 3.2.2]

Element	***Example Questions Describing the Element***
Rationale	*Why is the study being done?*
Purpose	*What is expected to be achieved with the study?*
The case	*Overall, what is being studied?*
Units of analysis	*In more detail, what is being studied?*
Theory	*What is the theoretical frame of reference?*
Research questions	*What knowledge will be sought or expected to be discovered?*
Propositions	*What particular (causal) relationships are to be investigated?*
Define concepts and measures	*How are entities and attributes being defined and measured?*
Methods of data collection	*How will data be collected?*
Methods of data analysis	*How will data be analyzed?*
Case selection strategy	*How will cases (and units of analyses) be identified and selected?*
Data selection strategy	*How will data be identified and selected? For example, who will be interviewed? What electronic data sources are available for use in the study? What nonelectronic, naturally occurring data sources are available for use in the study?*
Replication strategy	*Is the study intended to literally replicate a previous study, or theoretically replicate a previous study; or is there no intention to replicate?*
Quality assurance, validity and reliability	*How will the data collected be checked for quality? How will the analysis be checked for quality?*

Lisa 3 – Intervjuude läbiviimismeelespea

Sissejuhatus

- Intervjuu on diplomitöö jaoks tehtava juhtumiuuringu jaoks. Lõputöö teema on Deklaratiivset tarkvarajuurutamist abistava tööriista analüüs ja arendus
- Intervjuu võetakse linti mille alusel tehakse kokkuvõte. Tegemist on toorandmega ja seda lõputöös ei avalikustata. Küll aga tehakse sellest kokkuvõte, mis lisatakse diplomitöö lisadesse. Pärast kokkuvõtte koostamist antakse see intervjueeritavale ülevaatamiseks ja vajadusel kommentaaride lisamiseks.
- Intervjuu viiakse läbi avatud küsimustega *semi-structured* (e.k. pool struktureeritud) stiilis

Eesmärk

- Saada küsimustele vastused
- Saada teada kas ettepanekud või hüpoteesid peavad paika
- Saada andmed, mille analüüsist saadakse nõuded lõputöö raames valmivale tööriistale

Juhtumiuuringu küsimused ja ettepanekud/hüpoteesid

- Küsimus 1: Kuidas saavutatakse juurutusprotsessi eelsed rakenduste soovitud olekud praegu?
    - Ettepanek 1.1: Kui rakenduste soovitud olekute saavutamiseks kasutatakse deklaratiivseid vahendeid, siis rakendatakse GitOpsi.
    - Ettepanek 1.2: Kui kasutatakse GitOpsi, siis rakenduste soovitud olekute kirjeldamiseks ei ole ühtset tööriista.
- Küsimus 2: Kuidas võiks rakenduste soovitud olekuid saavutada ideaalis?
    - Ettepanek 2.1: Rakenduste soovitud olekuid võiks saavutada ühtse käsurea utiliidi abil.

- Ettepanek 2.2: Rakenduste soovitud olekuid soovitakse deklareerida kasutades deklaratiivseid vahendeid ja GitOpsi.

Lisa 4 – Üksikjuhtumi aruande koostamise juhend

Üksikjuhtumi aruande koostamise juhendi loomisel võeti eeskuju Hösti ja teiste [14, Ch. 5.3.2] ideedest. Alljärgnevalt on välja toodud kaks meetodit kuidas võiks aruande koostamisele läheneda. Meetodeid võib kasutada eraldiseisvalt või kombineerides.

Meetod 1

1. Koosta huvipakkuvate märksõnade kogum
2. Tööta andmetega ja kaardista need märksõnadega
3. Kasutades märksõnadega seotud materjali teosta järeldused:
    a. Võrdle andmeid erinevate märksõnadega
    b. Võrdle erinevaid märksõnu

Meetod 2

Developing a case description [19] ehk koonda andmed sektsioonidesse, lisa sektsioonidele iseloomustavad märksõnad ning teosta analüüs. Nimetatud protsess etappidena:

1. Koonda andmed sektsioonidesse
2. Lisa sektsioonidele neid iseloomustavad märksõnad
3. Analüüsi sektsioone ja märksõnu tervikuna ning tee nende põhjal järeldused

Lisa 5 – Üksikjuhtum 1 aruanne

Üksikjuhtumi aruanne koosneb kahest osast:

- Toorandmete ehk intervjuu ja koodi analüüsi alusel koostatud sektsioonidest ja neid iseloomustavatest märksõnadest
- Üksikjuhtumi kokkuvõttest

Üksikjuhtumi kokkuvõtte koostamisel lähtuti Yini [19] poolt kirjeldatud narratiivi formaadist, kus juhtumiuuring võetakse kokku seda kirjeldava ja analüüsiva jutustusena.

Sektsioonid ja nende märksõnad

Järgnevalt on välja toodud kolm tabelit, esimene ja viimane (Tabel 2. ja Tabel 4) on koostatud intervjuu andmete põhjal ja keskmine tabel (Tabel 3.) on koostatud juurutusprotsessi koodi analüüsi põhjal. Analüüsitud juurutusprotsessi kood oli kasutusel enne diplomitöö raames arendatud tööriista kasutuselevõtte. Esialgsed toorandmed on puhastatud, kokku võetud ning lisatud märksõnadega kõrvutatud sektsioonidesse. Selline lähenemine aitab paremini üldistada ja esile tuua juhtumit iseloomustavad tunnused.

Tabel 2. Juhtum 1 andmed ja märksõnad esialgsest lahendusest

Sektsioon	**Märksõna**
Funktsioon, mis tegeles uuendamisega ei saanud tegelikult asja sisust aru. Tegemist oli Perli käskude *search* ja *replace*-ga.	Manuaalne tegevus
Lahenduse funktsionaalsus oli piiratud - muudeti ainult kujutise märgist.	Piiratud funktsionaalsus
Kui oli vaja muuta Kubernetese *Update Strategy*-t, siis selle jaoks tehti eraldi Perli käsk.	Piiratud funktsionaalsus
Lahendus ei olnud standardne ja alati ei olnud veahaldust. Kui töötas, siis oli hästi, kui ei töötanud, siis pidi DevOps-ga tegelev inimene uurima mis on valesti.	Puuduv või kesine veahaldus
Uut CI/CD ahelat alustades oli kood küll taaskasutatav, kuid probleem oli selles, et uue ahela korral täiustati taaskasutatud koodi, kuid see täiustatud kood ei jõudnud esialgsetesse versioonidesse.	Puudulik versioonihaldus
CI/CD ahelates küll taaskasutati erinevatel ajahetkedel kirjutatud koodi, kuid neil puudus korrektne versioonihaldus.	Puudulik versioonihaldus

Hiljem kasutati jagatud teeki, kus muudatused küll jõudsid kõikidesse versioonidesse, aga olid aga omad puudused. Näiteks ei olnud väga põhjalikke kontrolle.	Puuduv või kesine veahaldus
Malli alusel faili koostamise lahendus oli primitiivne. Näiteks puudus korralik veahaldus.	Puuduv või kesine veahaldus

Tabel 3. Juhtum 1 andmed ja märksõnad analüüsitud koodist

Sektsioon	**Märksõna**
Lahendus oli Jenkinsi spetsiifiline.	Keskkonna spetsiifiline
Sed ja git tulemuste manuaalne kontrollimine.	Manuaalne tegevus
Liiga pikad funktsioonid – raskesti loetavad ja teevad liiga palju asju korraga.	Raskesti loetav
Liiga primitiivne või kohati olematu veahaldus .	Puuduv või kesine veahaldus
Lahendust oli ebamugav testida, kuna testida tuli manuaalselt ja testimiseks tuli käivitada CI/CD keskkond.	Ebamugav testimine

Tabel 4. Juhtum 1 andmed ja märksõnad tööriista soovitud omadustes

Sektsioon	**Märksõna**
Konkreetse otstarbega tarkvara.	Soovitud funktsionaalsus
Tarneahela kood oleks lühem ja lihtsamini arusaadav.	Soovitud funktsionaalsus
Peaks asendama vana lahenduse.	Taaskasutatav
Argo CD integratsioon, mis tähendab seda, et tööriista on võimalik kasutada keskkonnast sõltumatult.	Keskkonnast sõltumatu
Meeskondade vaheline töö ühtlustumine.	Ühtne
Kõik tööriista muudatused ja parendused on kõigile kättesaadavad.	Vabavaraline

Kokkuvõte

Üksikjuhtumi käigus prooviti vastus leida kahele küsimusele - milline on hetkel kasutusel olev lahendus ja milline on soovitud lahendus? Rakenduste soovitud olekute saavutamiseks kasutati vastavalt vajadusele kohandatud loogikat. Selline lähenemine oli küll rahuldav ja kui töötas, siis oli hästi, kuid sellel olid mitmed puudused.

Lahendusel puudus korralik veahaldus, ehk kui midagi läks valesti või kui juhtus midagi soovimatut, siis selle tuvastamine võis olla aeganõudev ja tülikas protsess. Lisaks puudus lahendusel korralik versiooni haldus, mis tähendas, et kohati võis kasutusel olla versioon, mida oli keeruline identifitseerida.

Lahendus tehnilisest perspektiivist vaadatuna täitis küll oma eesmärki, aga oli raskesti loetav ja testitav. Testimiseks oleks sobinud ainult *end-to-end* testid. Lahendus oli pigem ühe juurutusprotsessi vajadusi lahendav funktsioon, kui laialdasemalt kasutatav vahend.

Lisaks olemasoleva lahenduse kasutamisele uuriti intervjueeritavalt milline võiks olla lahendus, mis eemaldaks selle puudujäägid ja kitsaskohad. Vastav lahendus peaks olema kindla otstarbega tööriist, kuhu ette andes konkreetsed argumendid teostatakse soovitud tulemused. Lahendus peaks olema võimeline asendama olemasoleva lahenduse ja peaks olemas kasutatav tulevastes sarnastes olukordades. Laiema kasutuskonna ja meeskondade vahelise töö ühtsustamiseks võiks lahendus olla vabavaraline.

Lisa 6 – Üksikjuhtum 2 aruanne

Üksikjuhtumi aruanne koosneb kahest osast:

- Toorandmete ehk intervjuu ja koodi analüüsi alusel koostatud sektsioonidest ja neid iseloomustavatest märksõnadest
- Üksikjuhtumi kokkuvõttest

Üksikjuhtumi kokkuvõtte koostamisel lähtuti Yini [19] poolt kirjeldatud narratiivi formaadist, kus juhtumiuuring võetakse kokku seda kirjeldava ja analüüsiva jutustusena.

Sektsioonid ja nende märksõnad

Järgnevalt on välja toodud kolm tabelit, esimene ja viimane (Tabel 5. ja Tabel 7) on koostatud intervjuu andmete põhjal ja keskmine tabel (Tabel 6.) on koostatud juurutusprotsessi koodi analüüsi põhjal. Analüüsitud juurutusprotsessi kood oli kasutusel enne diplomitöö raames arendatud tööriista kasutuselevõtte. Esialgsed toorandmed on puhastatud, kokku võetud ning lisatud märksõnadega kõrvutatud sektsioonidesse. Selline lähenemine aitab paremini üldistada ja esile tuua juhtumit iseloomustavad tunnused.

Tabel 5. Juhtum 2 andmed ja märksõnad esialgsest lahendusest

Sektsioon	**Märksõna**
Rakenduste soovitud olekute saavutamine realiseeriti erinevate inimeste poolt isemoodi.	Puudulik versioonihaldus
Kasutati käsurea tööriistu nagu sed, grep, awk mille süntaksid kohati erinesid. See tegi loetavuse ja arusaamise ebaselgeks.	Mitmeti mõistetav
Konfiguratsiooni failide muutmiseks tuli kasutada kubectli käske.	Piiratud funktsionaalsus
Kubectli käsud olid versiooniti erinevad ja see tegi koodi raskesti hallatavaks.	Mitmeti mõistetav
Kui grepi või sed-ga midagi valesti tehti, siis tulemused olid ebameeldivd.	Puuduv või kesine veahaldus
Giti operatsioonid ja kubectli spetsiifiline konfiguratsioon tuli manuaalselt teostada.	Manuaalne tegevus

Tabel 6. Juhtum 2 andmed ja märksõnad analüüsitud koodist

Sektsioon	**Märksõna**
Sed ja giti läbiviimise manuaalne kontrollimine	Manuaalne tegevus
Lahendus oli Jenkinsi spetsiifiline.	Keskkonna spetsiifiline
Liiga primitiivne veahaldus.	Puuduv või kesine veahaldus
Manuaalne tulemuse testimine.	Ebamugav testimine

Tabel 7. Juhtum 2 andmed ja märksõnad tööriista soovitud omadustes

Sektsioon	**Märksõna**
Lahendus peaks aru saama konfiguratsioonidest ja olema võimeline vahetama kujutisi.	Soovitud funktsionaalsus
Muudatuste tegemiseks ei oleks vaja faile eraldi defineerida, vaid tööriist oskaks vastavalt asukohale need ise üles otsida .	Soovitud funktsionaalsus
Oleks mitmes projektis kasutatav ja ühesugune.	Ühtne
Lahendus peaks olema stabiilne ja töökindel.	Stabiilne
Lahendus võiks toetada Kubernetese juurutusstrateegia muutmist.	Soovitud funktsionaalsus
Lahendus oskab ühenduda Argo CD serveriga, pärast mida oskab käivitada sünkroniseerimise.	Soovitud funktsionaalsus
Kogu protsess võiks toimuda ühe sammuna – muutmine ja sünkroniseerimine.	Soovitud funktsionaalsus
Lahendus võiks olla juurutuskeskkonnast sõltumatu.	Keskkonnast sõltumatu

Kokkuvõte

Üksikjuhtumi läbiviimise käigus prooviti vastus leida kahele küsimusele - milline on hetkel kasutusel olev lahendus ja milline on soovitud lahendus? Intervjuu- ja koodianalüüsi põhjal selgus, et rakenduste soovitud olekute saavutamiseks kasutati vastavalt vajadusele kohandatud loogikat. Selline lähenemine oli küll rahuldav ja kui töötas, siis oli hästi, kuid sellel olid mitmed puudused.

Konfiguratsiooni muudatuste tegemiseks kasutati käsurea tööriistu, nagu sed, grep, või awk, mille süntaksid võisid vastavalt kasutajast erineda. Selline lähenemine tõi omajagu

probleeme, kuna see tegi lahenduse kasutamise ebamugavaks ja mitmeti mõistetavaks. See tähendas seda, et kui süntaksist tulenevatest erisustest midagi valesti mõisteti, siis tulemused olid enamasti soovimatud ja nendega oli vaja tegeleda.

Tänu koodianalüüsile saab veel välja tuua, et lahendus oli ühe keskkonna spetsiifiline ja sellel puudus korralik veahaldus.

Lisaks uuriti milline võiks olla parem viis uuritud ja tulevaste sarnaste situatsioonide lahendamiseks. Toodi välja, et lahendus peaks aru saama konfiguratsiooni failide sisust ja olema võimeline vahetama soovitud väärtuseid. Olgu selleks siis kas kujutise versiooni vahetamine või juurutusstrateegia uuendamine. Lisaks sooviti, et lahendus võiks olla keskkonnast sõltumatu töökindel ja stabiilne.

Lisa 7 – Üksikjuhtum 3 aruanne

Üksikjuhtumi aruanne koosneb kahest osast:

- Toorandmete ehk intervjuu ja koodi analüüsi alusel koostatud sektsioonidest ja neid iseloomustavatest märksõnadest
- Üksikjuhtumi kokkuvõttest

Üksikjuhtumi kokkuvõtte koostamisel lähtuti Yini [19] poolt kirjeldatud narratiivi formaadist, kus juhtumiuuring võetakse kokku seda kirjeldava ja analüüsiva jutustusena.

Sektsioonid ja nende märksõnad

Järgnevalt on välja toodud kolm tabelit, esimene ja viimane (Tabel 8 ja Tabel 10) on koostatud intervjuu andmete põhjal ja keskmine tabel (Tabel 9) on koostatud juurutusprotsessi koodi analüüsi põhjal. Analüüsitud juurutusprotsessi kood oli kasutusel enne diplomitöö raames arendatud tööriista kasutuselevõtte. Esialgsed toorandmed on puhastatud, kokku võetud ning lisatud märksõnadega kõrvutatud sektsioonidesse. Selline lähenemine aitab paremini üldistada ja esile tuua juhtumit iseloomustavad tunnused.

Tabel 8. Juhtum 3 andmed ja märksõnad esialgsest lahendusest

Sektsioon	**Märksõna**
Faili muudatuste tegemiseks kasutati kubectli.	Piiratud funktsionaalsus
Kubectl oli rahuldav, kui oli täpselt üks yaml ja üks kujutis mida tuli uuendada.	Piiratud funktsionaalsus
Kubectliga oli mitme konfiguratsiooni failide muudatuste tegemine tülikas. See nõudis vastava loogika defineerimist.	Ebamugav arendamine

Tabel 9. Juhtum 3 andmed ja märksõnad analüüsitud koodist

Sektsioon	**Märksõna**
Konfiguratsiooni muudatuste tegemiseks kasutati kubectl patchi.	Manuaalne tegevus
Puudus Giti integratsioon.	Manuaalne tegevus
Puudus korralik veahaldus.	Puuduv või kesine veahaldus

Tabel 10. Juhtum 3 andmed ja märksõnad tööriista soovitud omadustes

Sektsioon	**Märksõna**
Konfiguratsiooni failide juurasukohta võiks saada määrata kataloogi põhiselt.	Soovitud funktsionaalsus
Lahendus võiks olla stiilis käivita ja unusta ehk ta peaks olema stabiilne ka konkreetne	Soovitud funktsionaalsus
Lahendus võiks toetada valikulist Kubernetese ressursside uuendamist.	Soovitud funktsionaalsus

Kokkuvõte

Üksikjuhtumi läbiviimise käigus prooviti vastus leida kahele küsimusele - milline on hetkel kasutusel olev lahendus ja milline on soovitud lahendus? Intervjuu- ja koodianalüüsi põhjal selgus, et rakenduste soovitud olekute saavutamiseks kasutati vastavalt vajadusele kohandatud loogikat. Selline lähenemine oli küll rahuldav, kuid sellel olid omad puudused.

Konfiguratsioonimuudatuste tegemiseks kasutati kubectli *patch* funktsionaalsust, mis ühe faili muudatuste tegemisteks oli rahuldav lahendus, aga kui faile oli rohkem, siis tähendas see juba konkreetse loogika kirjeldamist.

Koodi analüüsist järeldus veel, et lahendusel puudus korralik veahaldus ning muudatuste deklareerimiseks tuli teostada vastavad giti toimingud.

Soovitud lahenduse omadustest tõi intervjueeritav välja, et konfiguratsiooni failide juurasukohta võiks saada määrata kataloogi põhiselt. Lisaks sooviti, et muudatuste ellu viimine võiks olla kuidagi Kubernetese ressurssidega määratav. Üleüldine soov oli, et lahendus võiks olla stabiilne ja veakindel.

Lisa 8 – Tarkvara nõuete spetsifikatsiooni mall

A software requirements specification template [24, Ch. 10]:

1. Introduction

1.1. Purpose

1.2. Document conventions

1.3. Project scope

1.4. References

2. Overall description

2.1. Product perspective

2.2. User classes and characteristics

2.3. Operating environment

2.4. Design and implementation constraints

2.5. Assumptions and dependencies

3. System features

3.x System feature X

3.x.1 Description

3.x.2 Functional requirements

4. Data requirements

4.1. Logical data model

4.2. Data dictionary

4.3. Reports

4.4. Data acquisition, integrity, retention, and disposal

5. External interface requirements

5.1. User interfaces

5.2. Software interfaces

5.3. Hardware interfaces

5.4. Communications interfaces

6. Quality attributes

6.1. Usability

6.2. Performance

6.3. Security

6.4. Safety

6.x [others]

7. Internationalization and localization requirements

8. Other requirements

Appendix A: Glossary

Appendix B: Analysis models

Lisa 9 – Käsk uuendamine funktsionaalsed nõuded

Tabel 11. Funktsionaalsed nõuded - uuendamine

Uuendamine	**Tööriistal peab olema uuendamise käsk**
.Logimine	Uuendamisel peab saama määrata logitaseme
.Git	Uuendamine peab võimaldama Giti toiminguteks vajaminevate argumentide sisestamist
.Hoidla	Uuendamine peab võimaldama Giti hoidla asukoha määramist
.Haru	Uuendamine peab võimaldama Giti haru asukoha määramist
.Võti	Uuendamine peab võimaldama SSH võtme asukoha määramist
.VõõrustajaKontroll	Uuendamine peab võimaldama StrictHostKeyChecking sisse- ja väljalülitamist
.ÜlesLaadimine	Uuendamine peab võimaldama Giti üleslaadimise sisse- ja väljalülitamist
.Autor	Uuendamine peab võimaldama Giti autori määramist
.Nimi	Autori määramisel peab olema võimalik sisestada autori nimi
.Meil	Autori määramisel peab olema võimalik sisestada autori meil
.Rakendus	Uuendamine peab võimaldama toimingute teostamise asukoha määramist

	.Asukoht	Uuendamine peab võimaldama toimingute teostamise asukoha määramist ühe argumendiga
	.Prefiks	Uuendamine peab võimaldama toimingute teostamise asukoha määramist osadena kasutades prefiksit
	.Nimeruum	Uuendamine peab võimaldama toimingute teostamise asukoha määramist osadena kasutades nimeruumi
	.Nimi	Uuendamine peab võimaldama toimingute teostamise asukoha määramist osadena kasutades nime
.Kujutised		Uuendamisel peab saama määrata uuendatavad kujutised
.JätaRegister		Uuendamisel peab olema võimallik kujutise registri osaga arvestamise sisse- ja väljalülitamine
.PaigaldamiseStrateegia		Uuendamisel peab olema võimalik valida paigaldamise strateegiat
.Rekursiivne		Uuendamine peab võimaldama rekursiivsuse, ehk muudatuste rakendamist kaustas ja tema alamkaustades, sisse- ja väljalülitamine

Lisa 10 – Käsk kopeerimine funktsionaalsed nõuded

Tabel 12. Funktsionaalsed nõuded - Kopeerimine

Kopeerimine	**Tööriistal peab olema kopeerimise käsk**
.Paigaldamine	Kopeerimine peab oskama modifitseerida dubleeritud konfiguratsioone vastavalt paigaldusfailile
.Logimine	Kopeerimisel peab saama määrata logitaseme
.Git	Kopeerimine peab võimaldama Giti toiminguteks vajaminevate argumentide sisestamist
.Hoidla	Kopeerimine peab võimaldama Giti hoidla asukoha määramist
.Haru	Kopeerimine peab võimaldama Giti haru asukoha määramist
.Võti	Kopeerimine peab võimaldama SSH võtme asukoha määramist
.VõõrustajaKontroll	Kopeerimisel peab saama StrictHostKeyChecking sisse- ja väljalülitamist
.ÜlesLaadimine	Kopeerimine peab võimaldama Giti üleslaadimise sisse- ja väljalülitamist
.Autor	Kopeerimine peab võimaldama Giti autori määramist
.Nimi	Autori määramisel peab olema võimalik sisestada autori nimi
.Meil	Autori määramisel peab olema võimalik sisestada autori meil

.Rakendus		Kopeerimine peab võimaldama toimingute teostamise asukoha määramist
	.Asukoht	Kopeerimine peab võimaldama toimingute teostamise asukoha määramist ühe argumendiga
	.Prefiks	Kopeerimine peab võimaldama toimingute teostamise asukoha määramist osadena kasutades prefiksit
	.Nimeruum	Kopeerimine peab võimaldama toimingute teostamise asukoha määramist osadena kasutades nimeruumi
	.Nimi	Kopeerimine peab võimaldama toimingute teostamise asukoha määramist osadena kasutades nime
	.Argo	Kopeerimine peab võimaldama toimingute teostamise asukoha määramist osadena kasutades Argo CD konfiguratsiooni asukohta
	.Yaml	Kopeerimine peab võimaldama toimingute teostamise asukoha määramist osadena kasutades yaml-konfiguratsioonide asukohta

Lisa 11 – Käsk kustutamine funktsionaalsed nõuded

Tabel 13. Funktsionaalsed nõuded - kustutamine

Kustutamine	**Tööriistal peab olema kustutamise käsk**
.Logimine	Kustutamisel peab saama määrata logitaseme
.Git	Kustutamisel peab võimaldama Giti toiminguteks vajaminevate argumentide sisestamist
.Hoidla	Kustutamine peab võimaldama Giti hoidla asukoha määramist
.Haru	Kustutamine peab võimaldama Giti haru asukoha määramist
.Võti	Kustutamine peab võimaldama SSH võtme asukoha määramist
.VõõrustajaKontroll	Kustutamisel peab saama StrictHostKeyChecking sisse- ja väljalülitamist
.ÜlesLaadimine	Kustutamine peab võimaldama Giti üleslaadimise sisse- ja väljalülitamist
.Autor	Kustutamine peab võimaldama Giti autori määramist
.Nimi	Autori määramisel peab olema võimalik sisestada autori nimi
.Meil	Autori määramisel peab olema võimalik sisestada autori meil
.Rakendus	Kustutamine peab võimaldama toimingute teostamise asukoha määramist

	.Asukoht	Kustutamine peab võimaldama toimingute teostamise asukoha määramist ühe argumendiga
	.Prefiks	Kustutamine peab võimaldama toimingute teostamise asukoha määramist osadena kasutades prefiksit
	.Nimeruum	Kustutamine peab võimaldama toimingute teostamise asukoha määramist osadena kasutades nimeruumi
	.Nimi	Kustutamine peab võimaldama toimingute teostamise asukoha määramist osadena kasutades nime
	.Argo	Kustutamine peab võimaldama toimingute teostamise asukoha määramist osadena kasutades Argo CD konfiguratsiooni asukohta
	.Yaml	Kustutamine peab võimaldama toimingute teostamise asukoha määramist osadena kasutades yaml-konfiguratsioonide asukohta

Lisa 12 – Käsk abi funktsionaalsed nõuded

Tabel 14. Funktsionaalsed nõuded - abi

Abi		**Tööriistal peab olema abistamise käsk**
	.Informatsioon	Abi peab andma informatsiooni kõikide tööriista poolt kasutatavate käskute ja nende sisendargumentide kohta

Lisa 13 – Tööriista sisendargumendid

Tabel 15. Tööriista sisendargumendid

Kirjeldav sisendargument	**Andmetüüp**	**Väärtus**
Logi tase	Tekst	Toodang või arendus
Giti repositooriumi aadress	Tekst	
Giti repositooriumi haru	Tekst	
SSH võtme asukoht	Tekst	Suhteline või absoluutine failitee
Võõrustaja kontroll	Tõeväärtus	Tõene või väär
Muudatuste üleslaadimine Giti	Tõeväärtus	Tõene või väär
Muutja nimi Giti repositoorimu logis	Tekst	
Muutja meil Giti repositoorimu logis	Tekst	
Võõrustaja kontroll	Tõeväärtus	Tõene või väär
Rakenduse failitee	Tekst	Absoluutne failitee Giti repositooriumi suhtes
Rakenduse osaline failitee - prefiks	Tekst	
Rakenduse osaline failitee - nimeruum	Tekst	
Rakenduse osaline failitee - nimi	Tekst	

Rakenduse osaline failitee - haru	Tekst	
Kujutised	Tekst	Näide: nimi:märgis, ...
Kujutise registri osaga arvestamine	Tõeväärtus	Tõene või väär
Paigaldamise strateegia	Tekst	RollingUpdate või Recreate
Rekursiivne muutmine	Tõeväärtus	Tõene või väär
Argo CD rakenduse failitee	Tekst	Suhteline failitee rakenduse failiteesuhtes
Yaml konfiguratsioonide asukoht	Tekst	Suhteline failitee rakenduse failiteesuhtes

Lisa 14 – Go-s kirjeldatud puu haru

```
// Node represents an element in the YAML document hierarchy. While documents
// are typically encoded and decoded into higher level types, such as structs
// and maps, Node is an intermediate representation that allows detailed
// control over the content being decoded or encoded.
//
// It's worth noting that although Node offers access into details such as
// line numbers, colums, and comments, the content when re-encoded will not
// have its original textual representation preserved. An effort is made to
// render the data plesantly, and to preserve comments near the data they
// describe, though.
//
// Values that make use of the Node type interact with the yaml package in the
// same way any other type would do, by encoding and decoding yaml data
// directly or indirectly into them.
//
// For example:
//
//     var person struct {
//             Name    string
//             Address yaml.Node
//     }
//     err := yaml.Unmarshal(data, &person)
//
// Or by itself:
//
//     var person Node
//     err := yaml.Unmarshal(data, &person)
//
type Node struct {
        // Kind defines whether the node is a document, a mapping, a sequence,
        // a scalar value, or an alias to another node. The specific data type of
        // scalar nodes may be obtained via the ShortTag and LongTag methods.
        Kind  Kind

        // Style allows customizing the apperance of the node in the tree.
        Style Style

        // Tag holds the YAML tag defining the data type for the value.
        // When decoding, this field will always be set to the resolved tag,
        // even when it wasn't explicitly provided in the YAML content.
        // When encoding, if this field is unset the value type will be
        // implied from the node properties, and if it is set, it will only
        // be serialized into the representation if TaggedStyle is used or
        // the implicit tag diverges from the provided one.
        Tag string

        // Value holds the unescaped and unquoted represenation of the value.
        Value string

        // Anchor holds the anchor name for this node, which allows aliases to point to it.
        Anchor string

        // Alias holds the node that this alias points to. Only valid when Kind is AliasNode.
        Alias *Node

        // Content holds contained nodes for documents, mappings, and sequences.
        Content []*Node

        // HeadComment holds any comments in the lines preceding the node and
        // not separated by an empty line.
        HeadComment string

        // LineComment holds any comments at the end of the line where the node is in.
        LineComment string

        // FootComment holds any comments following the node and before empty lines.
        FootComment string

        // Line and Column hold the node position in the decoded YAML text.
        // These fields are not respected when encoding the node.
        Line   int
        Column int
}
```

Joonis 5. Go-s kirjeldatud puu haru [40]