

TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Markus Peterson

Dockeri konteinerite kaughaldus IoT seadmetes

Bakalaureusetöö (9 EAP)

Juhendaja: Pelle Jakovits

Tartu 2018

Konteinerite kaughaldus IoT seadmetes

Lühikokkuvõte:

Käesoleva bakalaureusetöö põhiteemaks on Cumulocity liidese loomine Dockeri konteinerite haldamiseks Cumulocityga integreeritud seadmetel üle võrgu. Docker lihtsustab oluliselt rakenduste arendamist ja kasutuselevõttu tootmiskeskkonnas. Selle kasu on näha ka IoT valdkonnas, kus seadmete paljususe tõttu on keeruline hallata rakenduste sõltuvusi ja uute rakenduste versioonide kasutuselevõttu. Cumulocity platvorm pakub seadmete haldust, kuid praegusel hetkel ei ole leida liidest Dockeri konteinerite administreerimiseks. Kasutades Cumulocity platvormi luuakse liides IoT seadmetel Dockeri konteinerite haldamiseks. Täpsemalt peab liides võimaldama konteinerite elutsükli kontrollimist, konteinerite seadistamist, konteinerite paigaldamist.

Võtmesõnad:

IoT, Cumulocity, Node.js, AngularJS, Java, Docker

CERCS:

P170 (Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria))

Remote management of containers in IoT devices

Abstract:

The aim of this study is to create Cumulocity interface to remotely manage Docker containers on devices that are integrated into Cumulocity platform. Docker simplifies developing applications and deployment into production environment. It's benefits are also seen in IoT field where it is hard to maintain dependencies and new application versions on multiple devices. Cumulocity features device control, but currently has no interface to interact with Docker and it's containers. Using Cumulocity author creates device agent and webinterface that allows to interact with Docker through Cumulocity platform. This solution should support controlling lifecycle of containers, container configuration, container deployment.

Keywords:

IoT, Cumulocity, Node.js, AngularJS, Java, Docker

CERCS:

P170 (Computer science, numerical analysis, systems, control)

Sisukord

1	Sissejuhatus	4
1.1	Sissejuhatus	4
1.2	Lahendus	5
1.3	Ülevaade	6
2	Cumulocity tutvustus	7
2.1	Üldtutvustus	7
2.2	Põhirakendused	7
2.3	API-liides ja seadmega suhtlus	10
3	Docker'i tutvustus	13
3.1	Üldtutvustus	13
3.2	Docker'i süsteemipildid	13
3.3	Docker'i konteinerid	13
4	Praktiline osa	14
4.1	Nõuded	14
4.2	Arhitektuur	14
4.3	Cumulocity veebiliidese mooduli loomine	15
4.4	Cumulocity agendi loomine	23
4.5	Lahenduse testimine	31
5	Kokkuvõte	32
	Viidatud kirjandus	33
	II. Litsents	34

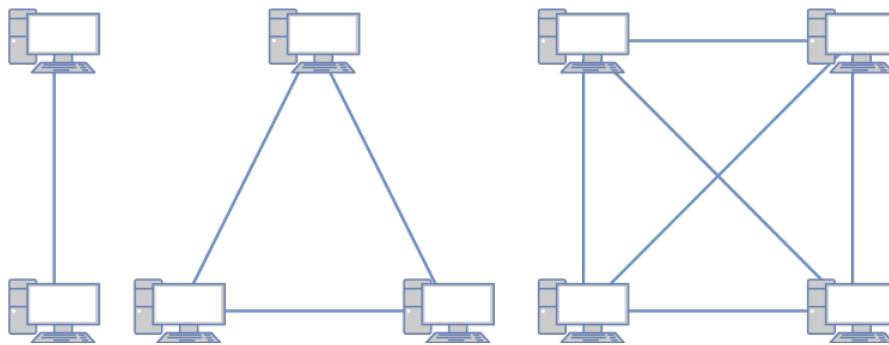
1 Sissejuhatus

1.1 Sissejuhatus

Asjade internet (inglise keeles Internet of Things) on igapäevaseadmetest koosnev infrastruktuur. See on tänapäeval üha rohkem populaarsemaks kujunev tehnoloogia, kuna ettevõtted näevad seda lisandväärtusena nii kasutajale kui ka ettevõttele endale. Seadmed asjade internetis on ühendatud internetiga ja vahetavad tihedalt andmeid ning integreeruvad üksteisega. Näitena saab tuua järgneva süsteemi.

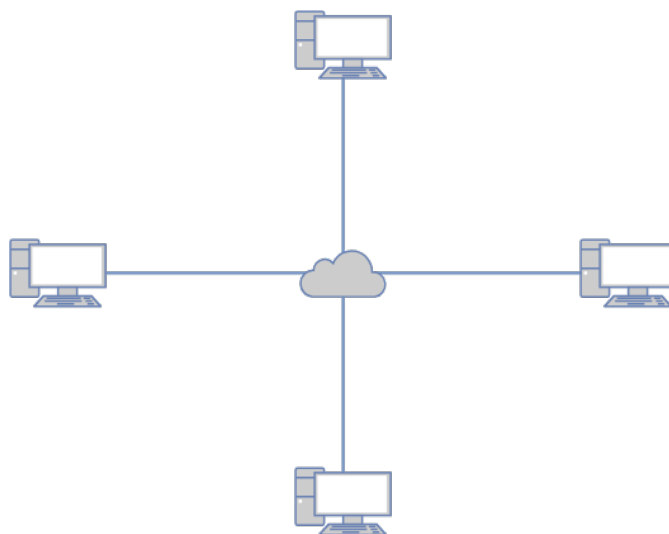
Matkaline jõuab külmal talvepäeval metsast koju ja soovib, et teda ootaks kodus ees kuum elektrisaun. Tal oli kogu retke jooksul käe peal nutikell, mis saab tema kehatemperatuuri jälgides aru, kui ta end läbi külmunult tunneb. Nutikell on samuti varustatud ka GPS seadmega ja tuvastab, kui kasutaja marsruudil koju on. Nende vaatluste järgi saadab nutikell kodus olevale elektrisaunale teate saun tööle panna.

Tavapärane lahendus oleks, et nutikella ja elektrisauna vahel on ühendus ja need suhtlevad otse üksteisega. Antud süsteemi laiendamisel seadmete lisamisega, näevad süsteemid erinevate seadmete arvu puhul välja sellised:



Joonis 1. Ühendused seadmete vahel tavapärasel süsteemis

Jooniselt 1 on näha, et seadmete lisamisel sellisesse süsteemi kasvab seadmetevaheline ühenduste arv kiiresti. Seetõttu on kasulik kaasata süsteemi keskseade, mis koondab endasse informatsiooni seadmete kohta ja teab, kuidas nendega suhelda. Teised seadmed saavad üksteisega suhtlemise asemel suhelda keskseadmega, mis võtab info saatjalt vastu ja edastab selle saaja kätte. Läbi keskseadme on seadmed paremini hallatavad, kuna on võimalik monitoorida seadmetel toimuvat. Sellist lahendust nimetatakse tsentraliseeritud süsteemiks.



Joonis 2. Ühendused seadmete vahel tsentraliseeritud süsteemis

Asjade interneti seadmete paremaks ühendamiseks, integreerimiseks ja haldamiseks loodi Cumulocity, mis töötab kui keskseade nende vahel. Cumulocity platvorm võimaldab luua paljudest erinevatest seadmetest koosnevaid süsteeme. Platvorm on eelkõige suunatud seadmete loojatele või rakenduste arendajatele, mistõttu tuleb luua sõltuvalt seadmest vastav liides platvormiga integreerumiseks. Selline geneerilisus lubab luua seadmele erivõimalustega funktsionaalsusi.

Seadme halduse juures möödapääsmatuks osaks on tarkvara haldus ja selle uuendamine. Enamasti igal rakendusel on sõltuvused, mis peavad olema installitud, et rakendus korrektselt töötaks. Asjade interneti seadmete paljususe tõttu on see mahukas töö. Seetõttu on kasulik kasutada Dockerit, mis on virtualiseerimistehnoloogia, mis konteinerdab rakenduse koos sõltuvuste ja konfiguratsiooniga. Dockeri eeliseks on ka väiksem ressursikasutus kui tavapärastel virtualiseerimistehnoloogiatel, kuna Docker kasutab sama kernelit nagu host masina kernel [7]. Integreerides Dockeri Cumulocity liidesesse, saab kontrollida seadmetel töötavaid rakendusi ning neid kergelt uuendada.

1.2 Lahendus

Pragusel hetkel ei ole leida lahendust, mis ühendaks omavahel Cumulocity platvormi ja Dockerit. Töö eesmärk on luua Cumulocity platvormi liides, mis lihtustaks Cumulocity platvormiga integreeritud seadmetel töötava tarkvara haldust kasutades Dockerit. Selle eeliseks on tarkvara paigutamine eelpakendatud modulaarsetesse tarkvarapakettidesse ehk süsteempiltidesse, mida saab lihtsasti liigutada ning käivitada seadmes, millesse on Docker installitud. Süsteempildi käivitamisel tehakse sellest konteiner, milles pakenda-

tud tarkvara töötab isoleeritult [7]. Kasutaja peab saama näha platvormi veebikeskkonnas seadmetel olevaid süsteemipilte ja konteinereid, mis on Lisaks peab saama süsteemipilte kustutada ja uusi alla laadida. Konteinereid peab saama käivitada, peatada ja kustutada. Samuti peab nägema hetkel aktiivseid konteinereid.

Liides luuakse kahes osas. Cumulocity kasutajaliidese plugin seadmete peal töötava Dockeri informatsiooni näitamiseks ning kontainerite ja süsteemipiltide haldamiseks luuakse Javascriptis kasutades Angular.js raamistikku. Teine osa on Cumulocity agent, mis vahendab infot Cumulocity platvormi ja seadmel töötava Dockeri tarkvara vahel. See luuakse Javas ja põhineb Cumulocity Bitbucketi repositooriumist pärit näitekoodil.

Lõpptulemuseks on agentrakendus ja veebiliidese moodul, mis on vabavaralised ehk igaüks võib vaadata ja kopeerida koodi enda kasutamise huvides. See võib olla abiks praegustele arendajatele, et lihtsustada asjade interneti seadmete tarkvara haldust või aitab tutvuda Cumulocity platvormiga.

1.3 Ülevaade

Ülejäänud lõputöö struktuur on järgnev. Teine peatükk tutvustab lähemalt Cumulocityt ja selle sisemist tööpõhimõtet. Kolmas peatükk räägib virtualiseerimistehnoloogiast Docker, selle süsteemipiltidest ja kontaineritest ning kuidas neid läbi Dockeri käsurea hallata. Neljas peatükk kirjeldab nõudeid loodavale Cumulocity liidesele, arhitektuurist, kuidas liides toimib, ja liidese loomisest endast nin lõpuks testitakse liidese vastavust püstitatud nõuetele. Viimane peatükk võtab lõputöö kokku.

2 Cumulocity tutvustus

2.1 Üldtutvustus

Cumulocity on peamiselt IoT seadmete integreerimiseks ja haldamiseks suunatud platvorm, mis ühendab seadmed ühisesse süsteemi ja võimaldab seadmetel suhelda üksteisega. Samuti lihtsustab see seadmetelt andmete kogumist, nende kontrollimist, konfigurimist ja pakub reaajalist logimist ning seadmete monitoorimist [6, 5]. Seadme registreerimine platvormi toimub Cumulocity agendi jooksutamisel seadmel ning seejärel tuleb seade aksepteerida platvormis endas. Agent on programm, mis töötab seadmel ja suhtleb Cumulocity REST API-liidesega ja teeb end kättesaadavaks platvormile. Agent määrab, milliseid operatsioone on võimalik selle seadmega teha ning milliseid andmeid see seade enda kohta saadab.

Cumulocity platvormi veebiliides on sama paindlik. Veebiliidest on võimalik laiendada AngularJS-l põhinevate moodulitena, mis võimaldab kohandada kasutajaliidese elemente seadme seisundi ja andmete kuvamiseks ning lisada interaktiivseid kasutajaliidese elemente operatsioonide välja kutsumiseks. Cumulocity on selleks valmistanud ka cumulocity-tools nimelise teegi, mis tagab uue mooduli baaskoodi ja teeb saadavaks Cumulocity põhifunktsioonide teegi, mille abil on Cumulocity platvormi REST API-liidesega kergem suhelda. Tagatud on ka kujundus, et moodulid sarnased välja näeksid.

Paindlikus on see, mis võimaldab paljudel seadmetel Cumulocity platvormiga integreeruda. Tuntumate seadmete nagu näiteks Arduino, Raspberry Pi, Tinkerforge, Kontron jaoks on loodud agendid, mida saab edasi arendada vastavalt vajadusele. Leidub ka Windowsile, Linuxile ja MacOSile mõeldud agente.

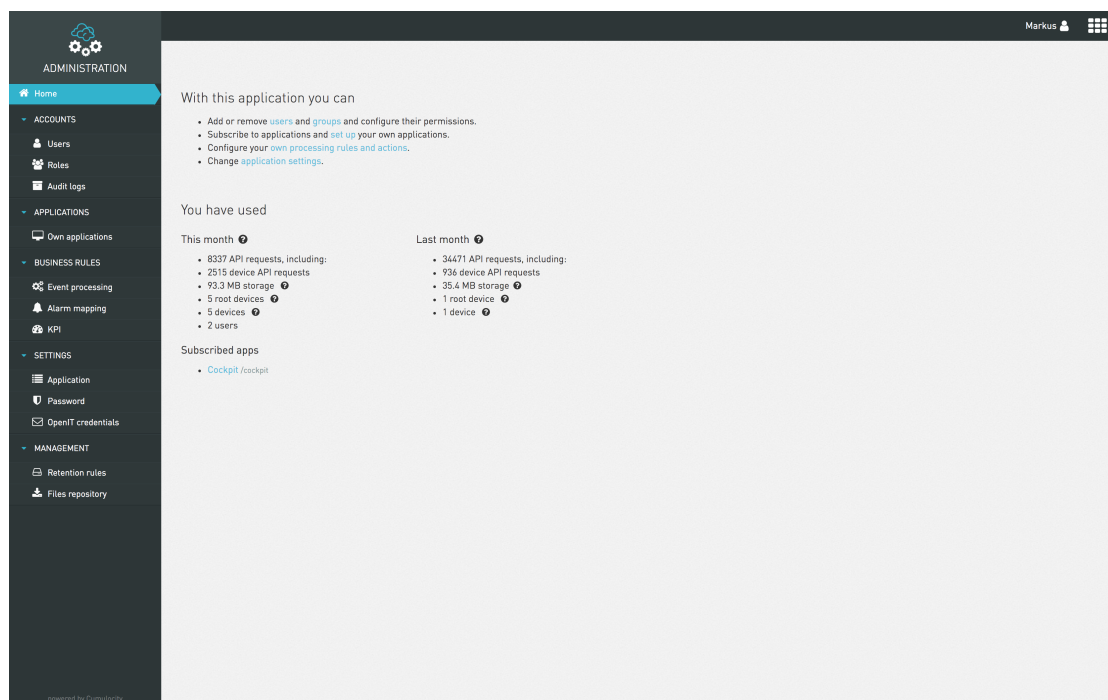
Cumulocity platvorm on üles ehitatud tenantite (eesti keeles üürnik) peale. Tenant kujutab endast kasutajate grupi üksust, läbi mille on võimalik platvormiga integreeruda. Sellised üksused on üksteisega seotud läbi puu struktuuri, mis tähendab, et igal tenantil on ülem, kellest see tenant tuletatud on, välja arvatud üks tenant. Selleks on juurtenant, mis on kuulub ettevõttele. Ülemtenantil on õigus alluvate tenantite haldamiseks enda õiguste piires. Tenantitel on samad õigused, mis nende ülemtenantil, kui ülemtenant pole teisiti määranud ja õiguseid kärpinud [2]. Tenantile määratud õigustest sõltub, mis selles tenantis on võimalik teha.

2.2 Põhirakendused

Cumulocity tenantis on esialgselt kolm põhirakendust, milledeks on administratsiooni, kopiti ja seadmehaldus rakendus.

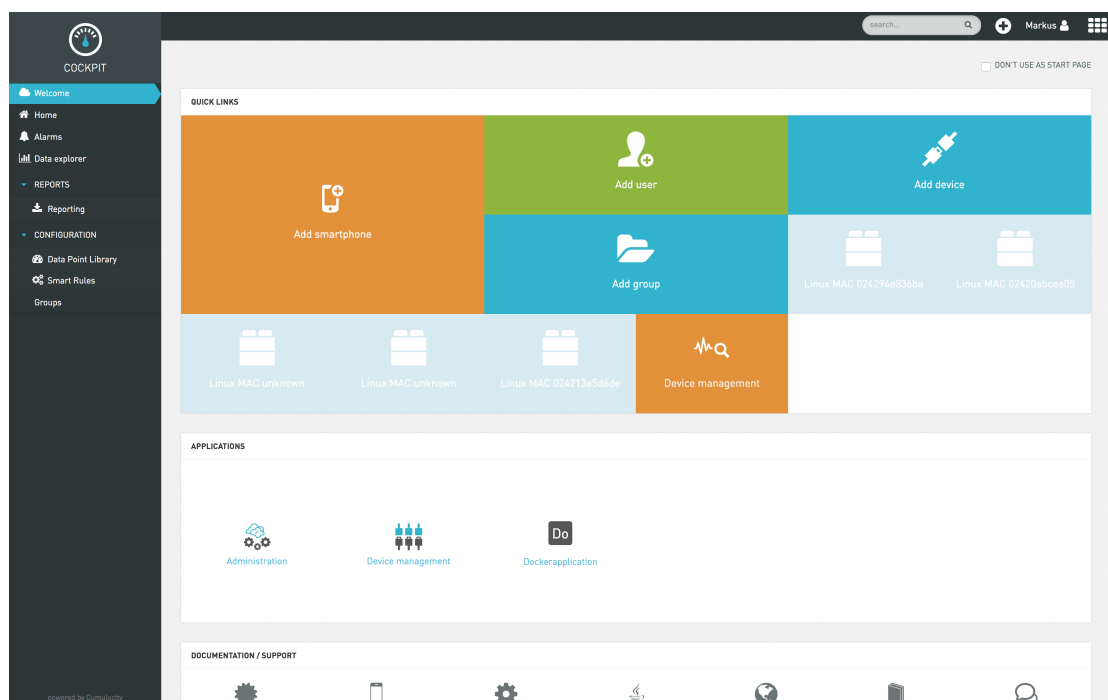
Administratsiooni rakenduse abil on võimalik lisada kasutajaid ja rolle ning neid

eemaldada; hallata tenanti rakendusi, nende seadistusi, konfiguratsioone ning andmete säilitamise reegleid; tenanti logi vaatamine; failide lisamine, alla laadimine ja kustutamine. Failide lisamise abil saab üles laadida uue veebiliidese mooduli zip failina. Põhirakendustes uusi mooduleid kasutada ei saa. Selleks tuleb neid kloonida administratsiooni rakenduse abil ja kasutada uusi mooduleid kloonitud rakendustes. Administratsiooni rakenduse esilehte on näha joonisel 3.



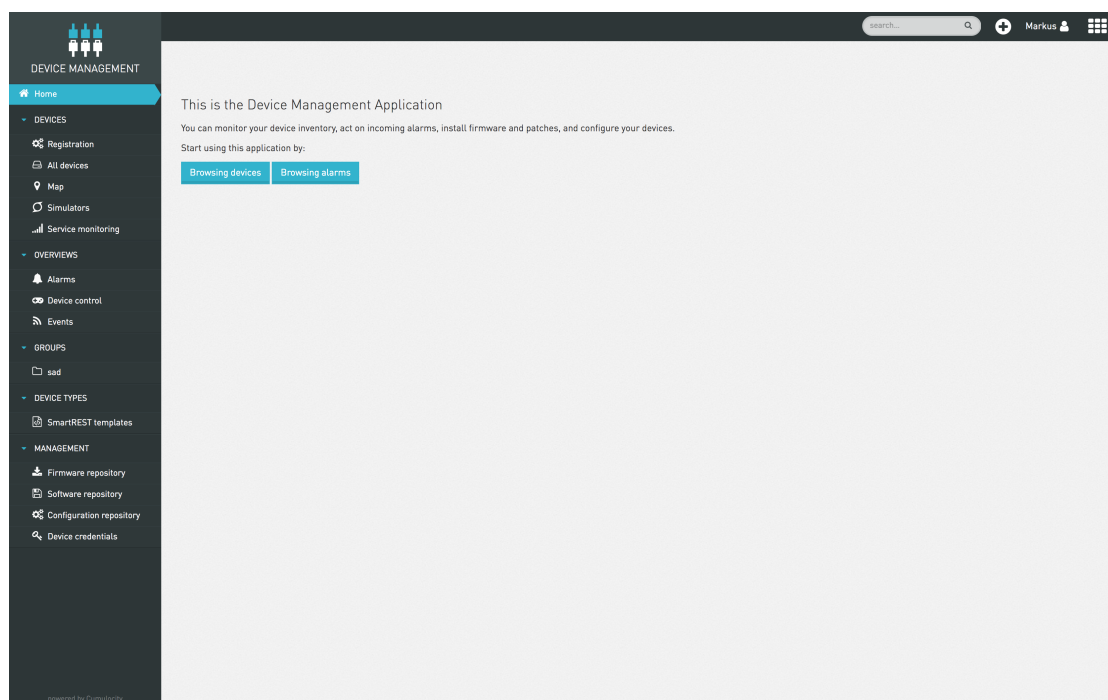
Joonis 3. Cumulocity administratsiooni rakendus

Kokpiti rakendus võimaldab saada kiiret ülevaadet kogu platvormi kohta. Võimalik korraga vaadata kõiki alarme ning neid hallata; luua raporte, neid kloonida ja kustutada; tenanti mõõtetulemusi lisada, kustutada ja visualiseerida. Võimalus luua esipaneele ning neid kohandada komponentidega. See võimaldab kuvada ülevaatlikku informatsiooni kõikide seadmete kohta. Komponente on võimalik valida paljude eeldefineeritute seast või juurde arendada. Kokpiti rakenduse esilehte on näha joonisel 4.



Joonis 4. Cumulocity kokpiti rakendus

Seadmehaldus rakenduse alt on võimalik teha erinevaid toiminguid seadmetega. Nendest tähtsaim on seadmete registreerimine. See toimub seadmel Cumulocity agendi käivitamisega mille järel ilmub dialoog veebiliideses, mille alt peab seadme registreerimist kinnitama. Sõltuvalt agendi implementatsioonist on vahepeal vajalik enne agendi käivitamist sisestada seadet identifitseeriv väärtus veebiliidesesse. Iga seadme kohta on võimalik vaadata üldist infot; konfiguratsiooni ning identifikaatoreid vaadata ja muuta; seadme geograafilise asukoha lisamine, muutmine ja jälgimine; seadme alarme vaadata ja kustutada; seadmega seotud sündmusi vaadata. Samuti on reaalse seadme puudumise korral võimalik luua simuleeritud seade ning neid hallata. Seadmehaldus rakenduse esilehte on näha joonisel 5.



Joonis 5. Cumulocity seadmehalduse rakendus

2.3 API-liides ja seadmega suhtlus

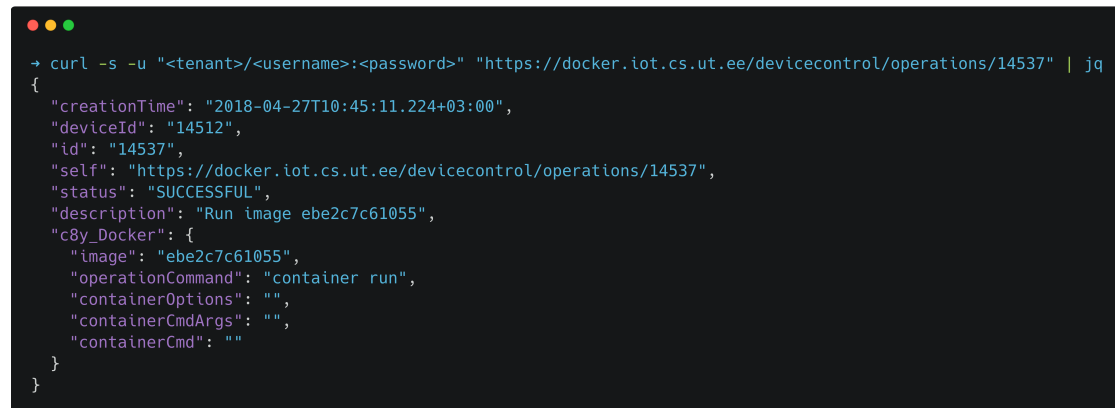
Cumulocityga suhtlus käib läbi andmelao, milles hoitakse kõik seadmetega seonduv info. Andmeladu jaguneb mõõdistusteks, sündmusteks, alarmideks, logideks ja operatsioonideks.

Mõõdistused koosnevad sensoritelt loetud numbrilistest andmetest näiteks temperatuuri väärtused või seadmete info põhjal arvutatud andmetest näiteks seadme teenuste kättesaadavus aja jooksul. Sündmusteks on kõik ülejäänud andmed, mis pärinevad sensoritelt, aga ei ole numbrilised väärtused näiteks ukse sensori päästmine. Sündmused võivad olla ka alarmid, kuid erinevalt sündmustest on alarmid prioriteetsemad ja annavad teavitusega kasutajale või süsteemi operaatorile teada, et kusagil esineb mingisugune viga. Alarmide teavitus püsib nii kaua, kui kasutaja või süsteemi operaator märgib alarmi lahendatuks. Operatsioonid tähistavad andmeid, mis on saadetud seadmele käivitamiseks või töötlemiseks [5]. Selleks võib olla näiteks teade elektrisaunale selle tööle panemiseks.

Seadmetel on võimalik kasutada andmelaoga suhtlemisel RESTful API (inglise keeles State Transfer Application Programming Interface). See on HTTP protokollil põhinev programm, mis järgib REST arhitektuuri stiili. Tehes päringuid API-liidesesse saavad seadmed pärida infot teiste seadmete kohta ja saata andmeid platvormi. API-dega

saab suhelda kindlas formaadis, mis sõltub API implementatsioonist [4].

Tänapäeval on Javascripti populaarsuse tõttu tavaks kujunenud kasutada JSON formaati (inglise keeles JavaScript Object Notation). JSON formaat kasutab inimloetavat teksti, et edastada andmeobjekte, milleks on serialiseeritav väärtus näiteks sõne, number, tõeväärtus, järjend, atribuudi ja väärtuse paaridest koosnev objekt või nullväärtus [1].



```
→ curl -s -u "<tenant>/<username>:<password>" "https://docker.iot.cs.ut.ee/devicecontrol/operations/14537" | jq
{
  "creationTime": "2018-04-27T10:45:11.224+03:00",
  "deviceId": "14512",
  "id": "14537",
  "self": "https://docker.iot.cs.ut.ee/devicecontrol/operations/14537",
  "status": "SUCCESSFUL",
  "description": "Run image ebe2c7c61055",
  "c8y_Docker": {
    "image": "ebe2c7c61055",
    "operationCommand": "container run",
    "containerOptions": "",
    "containerCmdArgs": "",
    "containerCmd": ""
  }
}
```

Joonis 6. Operatsiooni andmed Cumulocity platvormis JSON formaadis

Joonisel 6 toodud näites on tehtud curl¹ programmi abil GET päring vastu Cumulocity API-liidest. Päringu vastus on parema loetavuse mõttes formaaditud jq² programmi abil, mis lisab vastusesse taanded ja reavahetused. Päringu tulemuseks on JSON objekt, mis sisaldab informatsiooni seadmele saadetud operatsiooni kohta.

API-liides defineerib võimalikud aadressid, millele on võimalik päring teha, ja andmed, mis nendelt aadressidelt tagastatakse. Joonisel 6 toodud näites aadress, mille vastu päring tehti, on ettemääratud Cumulocity API-liidese poolt. REST API-liides implementerib tavaliselt 4 päringutüüpi [8]:

- GET - ressursist andmete pärimiseks
- POST - ressursis uute andmete loomiseks
- PUT - ressursis andmete uuendamiseks/muutmiseks
- DELETE - ressursis andmete kustutamiseks
- OPTIONS - ressursis toetatud operatsioonide pärimiseks

¹<https://curl.haxx.se/docs/manpage.html>

²<https://stedolan.github.io/jq/manual/v1.5/>

Cumulocity on teinud API-liidese võimalikult turvaliseks. API-liidese aadressid, mis peavad olema kaitstud kõrvaliste isikute eest, on turvatud primitiivse autentimisega (inglise keeles basic authentication). Nendele päringu tegemiseks on vajalik päringu päises kaasa saata kasutajanime ja parooli kombinatsioon, mis on omavahel ühendatud kooloniga ja kodeeritud base64 sõneks [4]. Joonisel 6 hoolitseb selle eest curl käsurea käsk, kuid allpool joonisel 7 on sellest pikem näide.



```
+ echo -n 'tenant/username:password' | base64
dGVuYW50L3VzZXJ1YXN3b3Jk

+ curl -s -H "Authorization: Basic dGVuYW50L3VzZXJ1YXN3b3Jk" "https://docker.iot.cs.ut.ee/devicecontrol/operations/14537" | jq
{
  "creationTime": "2018-04-27T10:45:11.224+03:00",
  "deviceId": "14512",
  "id": "14537",
  "self": "https://docker.iot.cs.ut.ee/devicecontrol/operations/14537",
  "status": "SUCCESSFUL",
  "description": "Run image ebe2c7c61055",
  "c8y_Docker": {
    "image": "ebe2c7c61055",
    "operationCommand": "container run",
    "containerOptions": "",
    "containerCmdArgs": "",
    "containerCmd": ""
  }
}
```

Joonis 7. Operatsiooni pärimine. Autentimis päise genereerimine

Täpsem info Cumulocity API-liidese kohta on leitav Cumulocity veebilehel paiknevast dokumentatsioonist, kus kirjeldatakse võimalike aadresse, millelt saab infot pärida, ja neilt tagastatavaid andmeid [4].

3 Dockeri tutvustus

3.1 Üldtutvustus

Docker on platvorm, mis pakub rakenduste arendamist ja juurutamist süsteempiltidena (inglise keeles image), mis kujutavad endast väiksemahulisi iseseisvaid käivitatavaid pakke, mis sisaldavad endas rakenduse koodi, käivituskeskkonda koos kõigi vajalike süsteemi tööriistade, rakenduse sõltuvustega ja seadistustega. Dockeri süsteempildid on võrreldavad lihtsustatud operatsioonisüsteemiga. Süsteempilte käivitades Dockeri abil luuakse konteiner, milles arendaja loodud rakendus jookseb.

Dockeri eeliseks on lihtne rakenduse produtseerimine, kuna see võimaldab jooksutada rakendust erinevate arvutite peal virtuaalselt samas keskkonnas. See teeb ka rakenduse arendamise lihtsamaks ja elimineerib arusaamatused, miks mõnes arvutis rakendus ei tööta.

Antud töös puutume kokku Dockeri andmetüüpidest konteinerite ja süsteempiltidega.

3.2 Dockeri süsteempildid

Dockeri süsteempildid on toorikud, millest on võimalik luua toimiv süsteem. Süsteempildi nimi pilt väljendub kujul "REPOSITORY:TAG", kus REPOSITORY on repositoorium või koodibaas, mida see süsteempilt representeerib ning TAG tähistab süsteempildi versiooni. Süsteempildi loomisel tähistatakse viimane versioon "latest" sümboliga, kui arendaja ei ole teisiti määranud. Süsteempilti saab jooksutada "docker run IMAGE" käsu abil, kus IMAGE on süsteempildi nimi ülaltoodud kujul. Süsteempildi versioon ei ole kohustuslik selle käsu jooksutamisel - puudumisel valitakse süsteemi pildi viimane versioon ehk "latest" versioon. Käsu tulemusena luuakse Dockeri konteiner, milles süsteempildis kirjeldatud süsteem töötab.

3.3 Dockeri konteinerid

Docker jooksutab protsesse isoleeritud konteinerites. Konteiner on protsess, mis jookseb peremeesarvutis (inglise keeles host). Peremeesarvutiks võib olla lokaalne arvuti või eemalasuv server. Kui kasutaja käivitab käsureal käsku "docker run", siis konteineris on protsess isoleeritud - tal on oma failisüsteem, oma juurdepääsuvõrk ja oma peremeesarvutist eraldiolev isoleeritud protsessipuu.

4 Praktiline osa

Kogu praktilises osas valminud kood on leitav Githubi repositooriumis aadressil https://github.com/marcus17777/bachelor_thesis.

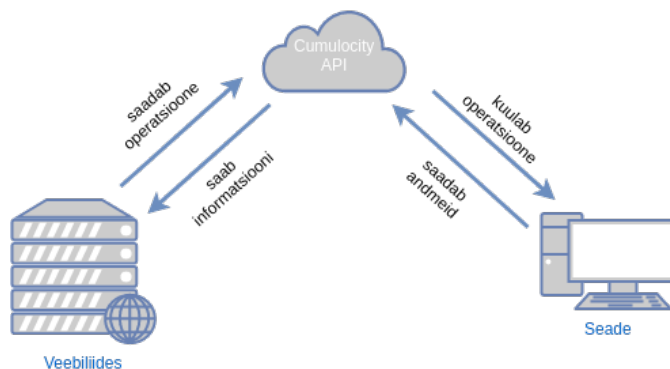
4.1 Nõuded

Loodav Cumulocity liides peab võimaldama saada ülevaatliku informatsiooni seadmel töötava Docker'i oleku kohta. Cumulocity veebilehel peab olema seadme juures olema vahekaart, mille alt võimalik näha seadmel olevaid süsteemipilte, mida on võimalik juurde lisada ja kustutada. Docker'i konteinerite jaoks on teine vahekaart, mille alt on võimalik näha seadmel olevaid konteinereid, neid käivitada, peatada ja kustutada. Samuti peab saama eristada töötavaid konteinereid seistvatest. Konteinereid peab saama sama vahekaardi peal luua süsteemipiltide järgi ja peab olema võimalus lisada argumente konteineri loomise juurde.

4.2 Arhitektuur

Cumulocity liides koosneb kahest osast: veebiliidese moodul, mille abil saab saata seadmele Docker'i spetsiifilisi operatsioone, ja agent, mis oskab vastu võtta neid operatsioone.

Veebiliidese moodul paikneb Cumulocity veebis ning suhtleb Cumulocity REST API-liidsega, kust saab infot seadmel oleva Docker'i süsteemipiltide ja konteinerite kohta ning saata seadmele operatsioone nende haldamiseks. Seadmel töötav agent kuulab pidevalt Cumulocity API-liidsest temale saadetud operatsioone ning saadab 5 sekundilise intervalli tagant andmeid Docker'i oleku kohta.



Joonis 8. Veebiliidese ja agendi suhtlus Cumulocity API-liidsega

Veebiliidese moodul luuakse programmeerimiskeeles Javascript, kasutades Node.js käituskeskonda. Node.js tuleb koos Npm paketi halduriga, mille abil saab installida

Cumulocity liidese arendamiseks vajalikud sõltuvused. Esimeseks on cumulocity-tools teek, mis tagab arendusserveri ja tööriistad, mille abil moodul luua. Selle installimise eelduseks on Node.js versioon 6.7 või uuem. Teiseks on Cumulocity UI pakett, mis teeb saadavaks visuaalsed komponendid, mille abil mooduli kasutajaliides üles ehitada. See tagab ühtse välimuse kogu Cumulocity veebilehel. Teegis on olemas ka meetodid Cumulocity andmelao ja seadmetega suhtlemise jaoks. Kolmandaks on Lodash, mis teeb Javascripti objektidega töötamise lihtsamaks.

Seadmel töötav agent luuakse Java programmeerimiskeeles. Autor otsustas agendi luua Cumulocity poolt valminud näitel, mis asub ettevõtte Bitbucketi repositooriumis [3]. Näide on loodud väljalülitatavate moodulite põhimõttel, mis tähendab, et Dockeri funktsionaalsusega agendi loomiseks piisab näidisagendile uue mooduli loomisest. Autor leiab, et see tagab loodava liidese skaleeritavuse, kui on vaja laiendada olemasolevat agendi Dockeri funktsionaalsusega. Agendi sõltuvusi haldab Maven, mis on Java projekti- ja paketi haldussüsteem.

4.3 Cumulocity veebiliidese mooduli loomine

Veebiliidese moodul peab võimaldama kasutajal saada ülevaade seadmel olevate Dockeri konteinerite ja süsteemipiltide kohta. Süsteemipilte ja konteinereid peab saama kustutada ning konteinereid peab saama ka käivitada, peatada ja uusi luua.

Veebiliidese mooduli loomiseks otsustati cumulocity-tools teegi poolt pakutavat arendusserverit, kuna see on Cumulocity poolt soovitatud viis veebiliidese mooduli arendamiseks. See võimaldab lokaalselt arendusmasinas testida loodavat moodulit. Seda saab käivitada käsurea käsuga "c8y server -u <tenant url>", kus <tenant url> on Cumulocity tenant veebiaadress.

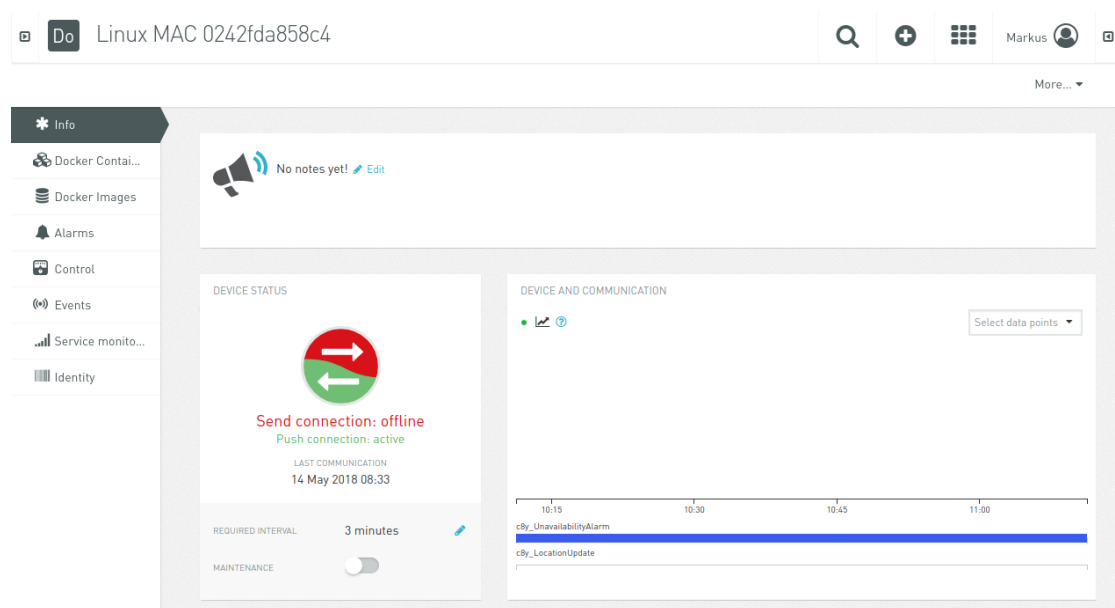
Järgnevalt tuleb moodul konfigureerida. Dockeri süsteemipiltide ja konteinerite kuvamiseks on vaja luua kaks vahekaarti Cumulocity seadmehaldus lehe peale. Seda saab teha c8yViewsProvider abil, mis on Cumulocity UI teegist pärinev abiklass. Joonisel 9 on näha veebiliidese mooduli konfiguratsiooni. Joonisel 10 on näha konfiguratsiooni tulemusena veebiliidesesse tekkivaid vahekaarte.

```

1  (function () {
2      'use strict';
3
4      angular
5          .module('myapp.dockerPlugin')
6          .config(configure);
7
8      /* @ngInject */
9      function configure(c8yViewsProvider) {
10         c8yViewsProvider.when('/device/:deviceId', {
11             name: 'Docker Containers',
12             icon: 'cubes',
13             priority: 1000,
14             templateUrl: ' :::PLUGIN_PATH:::/dockerContainers/views/index.html',
15             controller: 'dockerContainersCtrl'
16         });
17
18         c8yViewsProvider.when('/device/:deviceId', {
19             name: 'Docker Images',
20             icon: 'database',
21             priority: 1000,
22             templateUrl: ' :::PLUGIN_PATH:::/dockerImages/views/index.html',
23             controller: 'dockerImagesCtrl'
24         });
25     }
26
27 }());

```

Joonis 9. Veebiliidese mooduli konfiguratsioon



Joonis 10. Uued vahekaardid Cumulocity seadmehaldus lehel. Nähtavad vasakul ääres.

Veebiliidese moodul peab ise hoolitsema andmete küsimise eest platvormist. Cumulocity API pihta päringute tegemiseks saab samuti kasutada Cumulocity UI teegis olevaid abiklasse³. Seadmel oleva Dockeri oleku kätte saamiseks saab kasutada c8yDevices abiklassi, millega saab seadme platvormi sisese identifikaatori abil kätte seadme detailid. Nende seas on olemas ka informatsioon Dockeri süsteemipiltide ja konteinerite kohta.

```
11 | | | this.getData = function() {  
12 | | | | return c8yDevices.detail(deviceID).then(_.property('data.models_Docker'));  
13 | | | }
```

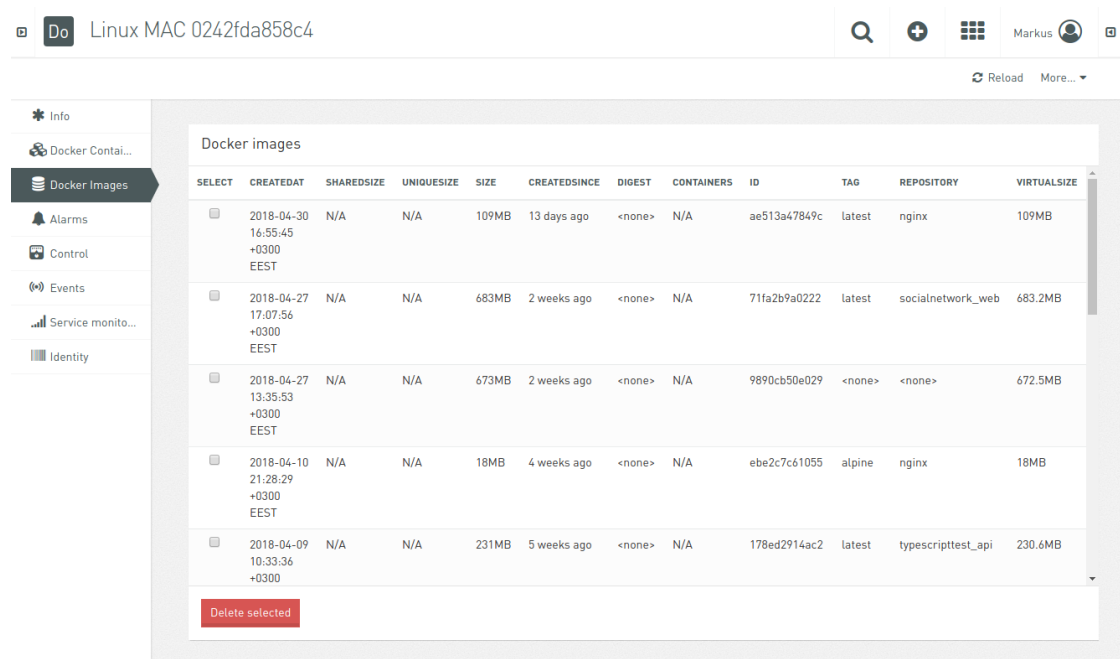
Joonis 11. Cumulocity API-st seadme Dockeri oleku pärimine

Docker'i süsteemipiltide kuvamiseks kasutatakse hüpertext-märgistuskeeles (HTML) kirjutatud malli, mida on näha joonisel 12. Selles on kasutatud Cumulocity UI poolt pakutavaid veebikomponente, mis tagab ühtse välimuse kogu ülejäänud platvormi veebiliidese. Andmed kuvatakse tabeliga "card" komponendi sees. Samuti on malli sees olemas juba nupp süsteemipiltide kustutamise operatsiooni saadmiseks. Malli tulemusena genereeritud kuva on näha joonisel 13.

```
1 | <div class="card">  
2 | | <div class="card-header separator">  
3 | | | <h4 class="card-title">Docker images</h4>  
4 | | </div>  
5 | | <div style="max-height: 500px; overflow: auto;">  
6 | | | <table class="table table-striped table-hover">  
7 | | | | <colgroup>  
8 | | | | </colgroup>  
9 | | | | <thead>  
10 | | | | | <tr>  
11 | | | | | | <th>Select</th>  
12 | | | | | | <th ng-repeat="header in headers">{{ header }}</th>  
13 | | | | | </tr>  
14 | | | | </thead>  
15 | | | | <tbody>  
16 | | | | | <tr ng-repeat="image in images">  
17 | | | | | | <td>  
18 | | | | | | | <label style="display: block;">  
19 | | | | | | | | <input style="margin: 0 auto;" class="checkbox" type="checkbox" ng-click="selector.select(image, $event.target.checked)"/>  
20 | | | | | | | </label>  
21 | | | | | | </td>  
22 | | | | | | <td ng-repeat="key in headers">{{ image[key] }}</td>  
23 | | | | | </tr>  
24 | | | | </tbody>  
25 | | | </table>  
26 | | </div>  
27 | | <div class="card-footer separator">  
28 | | | <button class="btn btn-danger" ng-click="deleteSelected()">  
29 | | | | Delete selected  
30 | | | </button>  
31 | | </div>  
32 | </div>
```

Joonis 12. Mall süsteemipiltide kuvamiseks

³<http://resources.cumulocity.com/documentation/jssdk/latest/api/c8y.core>



Joonis 13. Süsteemipiltide kuva Cumulocity seadmehaldus lehel

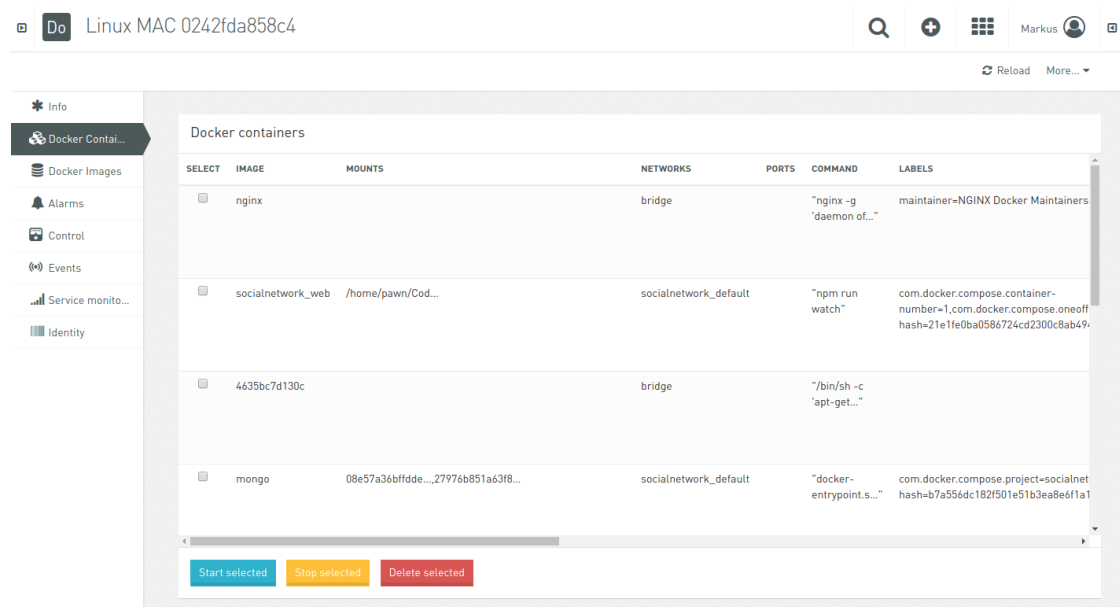
Docker konteinerite kuvamise puhul on mall sarnane ja näeb välja samasugune nagu süsteemipiltide puhul, kuid esineb kolm nuppu konteinerite käivitamise, peatamise ja kustutamise operatsioonide saatmiseks. Mall on nähtav joonisel 14 ja selle järgi genereeritud kuva joonisel 15.

```

9 <div class="card">
10 <div class="card-header separator">
11 <h4 class="card-title">Docker containers</h4>
12 </div>
13 <div style="max-height: 500px; overflow: auto;">
14 <table class="table table-striped table-hover">
15 <colgroup>
16 </colgroup>
17 <thead>
18 <tr>
19 <th>Select</th>
20 <th ng-repeat="header in headers">{{ header }}</th>
21 </tr>
22 </thead>
23 <tbody>
24 <tr ng-repeat="container in containers" ng-class="{ 'success': container.is_active}">
25 <td>
26 <label style="display: block;">
27 <input style="margin: 0 auto;" class="checkbox" type="checkbox" ng-click="selector.select(container.ID, $event.target.checked)"/>
28 </label>
29 </td>
30 <td ng-repeat="key in headers">{{ container[key] }}</td>
31 </tr>
32 </tbody>
33 </table>
34 </div>
35 <div class="card-footer separator">
36 <button class="btn btn-primary" ng-click="startSelected()">
37 Start selected
38 </button>
39 <button class="btn btn-warning" ng-click="stopSelected()">
40 Stop selected
41 </button>
42 <button class="btn btn-danger" ng-click="deleteSelected()">
43 Delete selected
44 </button>
45 </div>
46 </div>

```

Joonis 14. Mall konteinerite kuvamiseks



Joonis 15. Konteinerite kuva Cumulocity seadmehaldus lehel

Mallis esineb ka teine "card" komponent. See on süsteemipiltide järgi konteinerite loomise operatsiooni saatmiseks, mis sisaldab endas nelja välja: süsteemipildi valik; konteineri käivitamise argumendid; käsk, mida jooksutatakse konteineri sees; argumendid konteineri sees jooksutatava käsu jaoks. Mall on nähtav joonisel 16 ja selle järgi genereeritud kuva joonisel 17.

```
49 <div class="card">
50   <div class="card-header-actions separator">
51     <h4 class="card-title">Create container</h4>
52   </div>
53   <div>
54     <div class="card-block">
55       <div class="form-group">
56         <label>Image</label>
57         <select required class="form-control monospaced" ng-model="form.image">
58           <option ng-repeat="image in images" value="{{ image.ID }}">
59             {{ image.ID }} {{ image.repository }}:{{ image.tag }}
60           </option>
61         </select>
62       </div>
63       <div class="form-group">
64         <label>Run options</label>
65         <input class="form-control" type="text" ng-model="form.runOptions">
66       </div>
67       <div class="form-group">
68         <label>Container command</label>
69         <input class="form-control" type="text" ng-model="form.containerCommand">
70       </div>
71       <div class="form-group">
72         <label>Container command args</label>
73         <input class="form-control" type="text" ng-model="form.containerCommandArgs">
74       </div>
75     </div>
76   </div>
77   <div class="card-footer separator">
78     <button class="btn btn-primary" ng-click="createContainer()">
79       Create
80     </button>
81   </div>
82 </div>
83
84 <style>
85   .monospaced {
86     font-family: Consolas, Monaco, Lucida Console, Liberation Mono, DejaVu Sans Mono, Bitstream Vera Sans Mono, Courier New, monospace;
87   }
88 </style>
```

Joonis 16. Mall konteinerite käivitamiseks

Joonis 17. Konteinerite loomise kuva Cumulocity seadmehaldus lehel

Et mallidel olevad nupud operatsioone saadaks, on vaja luua meetodid, mis teevad päringud Cumulocity API-liidesesse. Selle kaudu operatsioonide loomiseks on Cumulocity UI teegis olemas `c8yDeviceControl` abiklass. Operatsiooni loomiseks on sellel "create", mis võtab argumendiks Javascripti objekti seadme identifikaatori, operatsiooni kirjeldusega ning ülejäänud osa on operatsiooni argumendid paaridena, kus paari esimene pool on seadme agendi poolt toetatud mooduli nimi ja paari teine pool on sellele moodulile saadetavad argumendid. Joonisel 18 on näide argumentidest.

```
c8yDeviceControl.create({
  deviceId: 123,
  description: "Stop containers 5e53c3f0283d",
  c8y_Docker: {
    operationCommand: "container stop",
    containerIDs: ['5e53c3f0283d']
  }
});
```

Joonis 18. Näide `c8yDeviceControl` abil operatsiooni saatmisest

Autor valis agendi Docker'i mooduli nimeks "c8y_Docker". Et moodul teaks, mida operatsiooni kätte saamisel teha, määrame moodulile saadetavate argumentide sisse muutuja "operationCommand". Selle väärtused peavad olema veebiliidese mooduli ja agendi vahel kooskõlas. Kui veebist saadetakse operatsioon "container stop", siis agendi moodul peab teadma, mida sellise operatsiooniga teha. Seega määrame järgmised konstandid:

- image rm - Süsteempiltide kustutamine

- container rm - Konteinerite kustutamine
- container stop - Konteinerite peatamine
- container start - Konteinerite käivitamine
- container run - Konteinerite loomine

Kasutades ülalmainitud konstante, luuakse joonistel 19, 20, 21, 22, 23 näidatud meetodid, mis saadavad Cumulocity API-liidesesse operatsiooni. Need meetodid kutsutakse välja joonistel 12, 14, 16 kirjeldatud nuppude vajutamisel.

```
this.deleteImages = function(IDs) {
  return c8yDeviceControl.create({
    deviceId: deviceID,
    description: "Delete images " + IDs.toString(),
    c8y_Docker: {
      operationCommand: "image rm",
      imageIDs: IDs
    }
  });
}
```

Joonis 19. Meetod süsteempiltide kustutamise operatsiooni loomiseks

```
this.deleteContainers = function(IDs) {
  return c8yDeviceControl.create({
    deviceId: deviceID,
    description: "Delete containers " + IDs.toString(),
    c8y_Docker: {
      operationCommand: "container rm",
      containerIDs: IDs
    }
  });
}
```

Joonis 20. Meetod konteinerite kustutamise operatsiooni loomiseks

```
this.stopContainers = function(IDs) {
  return c8yDeviceControl.create({
    deviceId: deviceID,
    description: "Stop containers " + IDs.toString(),
    c8y_Docker: {
      operationCommand: "container stop",
      containerIDs: IDs
    }
  });
}
```

Joonis 21. Meetod konteinerite peatamise operatsiooni loomiseks

```
this.startContainers = function(IDs) {
  return c8yDeviceControl.create({
    deviceId: deviceID,
    description: "Start containers " + IDs.toString(),
    c8y_Docker: {
      operationCommand: "container start",
      containerIDs: IDs
    }
  });
}
```

Joonis 22. Meetod konteinerite käivitamise operatsiooni loomiseks

```

this.createContainer = function(containerOptions, image, containerCmd, containerCmdArgs) {
  if (_.isEmpty(image)) {
    throw new Error("Must specify image");
  }

  return c8yDeviceControl.create({
    deviceId: deviceId,
    description: "Run image " + image,
    c8y_Docker: {
      operationCommand: "container run",

      containerOptions: containerOptions || "",
      image: image,
      containerCmd: containerCmd || "",
      containerCmdArgs: containerCmdArgs || ""
    }
  })
}

```

Joonis 23. Meetod konteinerite loomise operatsiooni loomiseks

4.4 Cumulocity agendi loomine

Agendi ülesanne on saada iga viie sekundi tagant Cumulocity andmelattu andmed Docker'i süsteemipiltide ja konteinerite kohta. Agent peab oskama kuulata operatsioonide sündmusi Cumulocity andmelaos ja seejärel vastavalt operatsioonile süsteemipiltide või konteinereid kustutada või konteinereid käivitada, peatada, uusi luua.

Loodav agent põhineb Cumulocity näidisagendi koodil, millele luuakse moodul Docker'i spetsiifiliste funktsioonide jaoks. Seadmel oleva Dockeriga suhtlemiseks kasutatakse Docker'i käsurea programmi. Selle jaoks loodi DockerRepository nimeline klass, mille abil käivitatakse Java koodi seest Docker'i käsurea programmi käsklusi. Selle jaoks on klassi sees asuv executeDockerCommand nimeline meetod.

```

private BufferedReader executeDockerCommand(String dockerCommand) {
    String[] command = new String[] {
        "/bin/bash",
        "-c",
        "docker " + dockerCommand
    };

    try {
        logger.debug(String.format("Executing command: %s", Arrays.toString(command)));
        Process process = Runtime.getRuntime().exec(command);

        BufferedReader stdout = new BufferedReader(new InputStreamReader(process.getInputStream()));
        BufferedReader stderr = new BufferedReader(new InputStreamReader(process.getErrorStream()));

        String error = stderr.lines().collect(Collectors.joining("\n"));
        if (error.length() > 0) {
            throw new RuntimeException(error);
        }

        return stdout;
    } catch (IOException e) {
        e.printStackTrace();
        return null;
    }
}

```

Joonis 24. Meetod DockerRepository klassis, mis käivitab Dockeri käsurea programmi

Docker'i süsteemipiltide ja konteinerite saamiseks on vaja kõigepealt defineerida klassid, milles neid andmeid hoida.


```

public class DockerImage {
    private String Containers;
    private String CreatedAt;
    private String CreatedSince;
    private String Digest;
    private String ID;
    private String Repository;
    private String SharedSize;
    private String Size;
    private String Tag;
    private String UniqueSize;
    private String VirtualSize;

    public String getContainers() { return Containers; }
    public String getCreatedAt() { return CreatedAt; }
    public String getCreatedSince() { return CreatedSince; }
    public String getDigest() { return Digest; }
    public String getID() { return ID; }
    public String getRepository() { return Repository; }
    public String getSharedSize() { return SharedSize; }
    public String getSize() { return Size; }
    public String getTag() { return Tag; }
    public String getUniqueSize() { return UniqueSize; }
    public String getVirtualSize() { return VirtualSize; }
}

```

Joonis 25. Klass Dockeri süsteempiltide hoidmiseks

```

public class DockerContainer {
    private String Command;
    private String CreatedAt;
    private String ID;
    private String Image;
    private String Labels;
    private String LocalVolumes;
    private String Mounts;
    private String Names;
    private String Networks;
    private String Ports;
    private String RunningFor;
    private String Size;
    private String Status;

    public String getCommand() { return Command; }
    public String getCreatedAt() { return CreatedAt; }
    public String getID() { return ID; }
    public String getImage() { return Image; }
    public String getLabels() { return Labels; }
    public String getLocalVolumes() { return LocalVolumes; }
    public String getMounts() { return Mounts; }
    public String getNames() { return Names; }
    public String getNetworks() { return Networks; }
    public String getPorts() { return Ports; }
    public String getRunningFor() { return RunningFor; }
    public String getSize() { return Size; }
    public String getStatus() { return Status; }
}

```

Joonis 26. Klass Dockeri konteinerite hoidmiseks

```

public class Docker {
    private List<DockerImage> images;
    private List<DockerContainer> containers;

    public List<DockerImage> getImages() {
        return images;
    }

    public void setImages(List<DockerImage> images) {
        this.images = images;
    }

    public List<DockerContainer> getContainers() {
        return containers;
    }

    public void setContainers(List<DockerContainer> containers) {
        this.containers = containers;
    }
}

```

Joonis 27. Klass Dockeri süsteempiltide ja konteinerite ühiseks hoiustamiseks

Joonisel 24 olevast meetodist on tuletatud DockerRepository klassi sisse meetodid, mille abil saab kätte seadmel olevad Dockeri süsteempildid ja konteinerid.

```
public List<DockerImage> getImages() {  
    return formatOutput(executeDockerCommand("image ls --format '{{json .}}'", DockerImage.class);  
}
```

Joonis 28. Meetod DockerRepository klassis, mis küsib Dockerilt süsteempildid

```
public List<DockerContainer> getContainers() {  
    return formatOutput(executeDockerCommand("container ls -a --format '{{json .}}'", DockerContainer.class);  
}
```

Joonis 29. Meetod DockerRepository klassis, mis küsib Dockerilt konteinerid

```
public Docker getState() {  
    Docker dockerModel = new Docker();  
    dockerModel.setImages(getImages());  
    dockerModel.setContainers(getContainers());  
    return dockerModel;  
}
```

Joonis 30. Meetod DockerRepository klassis, mis tagastab Dockeri oleku

Dockeri käsurea programmi on nendes käivitatud "--format '{{json .}}'" lipuga, kuna JSON formaati on lihtne töödelda. Selleks on kasutatud com.google.gson teegist Gson klassi.

```

private <T> List<T> formatOutput (BufferedReader stream, Class<T> klass) {
    Gson g = new Gson();
    return stream.lines().map((s -> g.fromJson(s, klass))).collect(Collectors.toList());
}

```

Joonis 31. Meetod DockerRepository klassis, mis formaadib Dockeri käsurea programmi väljundi

Mooduli keskmeks on DockerDriver nimeline klass, mille ülemklassiks on c8y.lx.driver.PollingDriver abstraktne klass, mis pärineb näidiskoodist. See implementeerib java.lang.Runnable liidest, mistõttu on vaja DockerDriver klassis defineerida "run" meetod. PollingDriver kutsus seda meetodit ajalise intervalli tagant välja, milleks on määratud DockerDriver klassis 5 sekundit. Selle meetodi abil saadetakse Cumulocity andmelattu informatsiooni Dockeri oleku kohta, mis saadakse DockerRepository klassi getState meetodi abil.

```

private void updateState() {
    ManagedObjectRepresentation mo = new ManagedObjectRepresentation();
    mo.setId(gid);
    mo.set(dockerRepository.getState());
    inventory.update(mo);
}

@Override
public void run() {
    System.out.println("DockerDriver polling");
    updateState();
}

```

Joonis 32. Meetod run DockerDriver klassis, mis saadab informatsiooni Dockeri oleku kohta

Joonisel 32 updateState meetodis luuakse uus andmelao objekt ManagedObjectRepresentation klassi järgi ning selle identifikaatoriks seatakse praeguse seadme identifikaator. Selle järgi teab Cumulocity andmeladu, millise seadme juurde see objekt kuulub. Seejärel uuendatakse andmeladu selle objektiga. Selle tulemusena on seadme Dockeri süsteemi-pildid ja konteinerid platvormis kätte saadavad ja veebiliideses kuvatavad.

Agendi operatsioonide kuulamine on juba Cumulocity näidisagendis ette defineeritud. Agent kuulab operatsioonide sündmusi andmelaos, mis sinna läbi API-liidesesse saadetakse, ja operatsiooni saabumisel saadetakse see agendi moodulitele laiali. Moodul peab seejärel aru saama, kas operatsioon oli talle suunatud või mõnele muule moodulile. Kuna agent agent kuulab kõiki operatsioone andmelaos, siis peab moodul otsustama ka, kas operatsioon on mõeldud sellele seadmele või mitte.

Selleks, et agent teaks, et moodul toetab operatsioonide käivitamist, peab DockerDriver klass implementeerima `c8y.lx.driver.OperationExecutor` liidese, mis nõuab, et moodul teataks oma operatsiooni tüübi, milleks on "c8y_Docker" nagu joonise 18 juures kirjeldatud on. Selleks peab defineerima kaks meetodit:

```
@Override
public OperationExecutor[] getSupportedOperations() {
    return new OperationExecutor[] { this };
}

@Override
public String supportedOperationType() {
    return "c8y_Docker";
}
```

Joonis 33. Meetodid Dockerdriveris, mis teatavad toetatud operatsiooni tüübi agendile

`OperationExecutor` liides ka nõuab "execute" meetodi defineerimist. Selles meetodis peab kontrollima, kas operatsioon on mõeldud sellele seadmele. Veebiliideses saadetud Dockeri operatsioonid sisaldavad endas "operationCommand" muutujat nagu joonise 18 juures kirjeldatud on. Selle järgi saab "execute" meetodis teada, millist Dockeri käsurea käsku jooksutada.

```

@Override
public void execute(OperationRepresentation operation, boolean cleanup) throws Exception {
    if (!gid.equals(operation.getDeviceId())) {
        // Silently ignore the operation if it is not targeted to us,
        // another driver will (hopefully) care.
        return;
    }

    if (cleanup) {
        operation.setStatus(OperationStatus.SUCCESSFUL.toString());
    } else {
        HashMap c8y_docker = (HashMap) operation.get("c8y_Docker");
        String operationCommand = (String) c8y_docker.get("operationCommand");

        switch (operationCommand) {
            case "image rm":
                for (Object id : ((ArrayList) c8y_docker.get("imageIDs"))) {
                    dockerRepository.deleteImage((String) id);
                }
                operation.setStatus(OperationStatus.SUCCESSFUL.toString());
                break;

            case "container rm":
                for (Object id : ((ArrayList) c8y_docker.get("containerIDs"))) {
                    dockerRepository.deleteContainer((String) id);
                }
                operation.setStatus(OperationStatus.SUCCESSFUL.toString());
                break;

            case "container start":
                for (Object id : ((ArrayList) c8y_docker.get("containerIDs"))) {
                    dockerRepository.startContainer((String) id);
                }
                operation.setStatus(OperationStatus.SUCCESSFUL.toString());
                break;

            case "container stop":
                for (Object id : ((ArrayList) c8y_docker.get("containerIDs"))) {
                    dockerRepository.stopContainer((String) id);
                }
                operation.setStatus(OperationStatus.SUCCESSFUL.toString());
                break;

            case "container run":
                String image = (String) c8y_docker.get("image");
                String options = "-d " + c8y_docker.get("containerOptions");
                String containercmd = (String) c8y_docker.get("containerCmd");
                String containerCmdArgs = (String) c8y_docker.get("containerCmdArgs");
                dockerRepository.runContainer(options, image, containercmd, containerCmdArgs);
                operation.setStatus(OperationStatus.SUCCESSFUL.toString());
                break;
        }
        updateState();
    }
}

```

Joonis 34. Meetod execute DockerDriver klassis, mis käivitab vastava Dockeri käsurea programmi sõltuvalt operatsioonist

Lõpetuseks vaja täiendada ka DockerRepository klassi süsteemipiltide kustutamise, konteinerite käivitamise, peatamise, kustutamise ja loomise meetoditega.

```
public void deleteImage(String id) {  
    executeDockerCommand(String.format("image rm %s", id));  
}
```

Joonis 35. Meetod DockerRepository klassis, mis käivitab Dockeri käsurea programmi süsteemipiltide kustutamiseks

```
public void deleteContainer(String id) {  
    executeDockerCommand(String.format("container rm %s", id));  
}
```

Joonis 36. Meetod DockerRepository klassis, mis käivitab Dockeri käsurea programmi konteinerite kustutamiseks

```
public void stopContainer(String id) {  
    executeDockerCommand(String.format("container stop %s", id));  
}
```

Joonis 37. Meetod DockerRepository klassis, mis käivitab Dockeri käsurea programmi konteinerite peatamiseks

```

public void startContainer(String id) {
    executeDockerCommand(String.format("container start %s", id));
}

```

Joonis 38. Meetod DockerRepository klassis, mis käivitab Dockeri käsura programmi konteinerite käivitamiseks

```

public void runContainer(String options, String image, String containerCmd, String containerCmdArgs) {
    executeDockerCommand(String.format("container run %s %s %s %s", options, image, containerCmd, containerCmdArgs));
}

```

Joonis 39. Meetod DockerRepository klassis, mis käivitab Dockeri käsura programmi konteinerite loomiseks

4.5 Lahenduse testimine

5 Kokkuvõte

Käesoleva bakalaureusetöö eesmärgiks oli luua liides, mis ühendab omavahel Cumulocity platvormi ja Dockeri. Liides pidi võimaldama saada ülevaade Dockeri süsteemipiltide ja konteinerite kohta. Süsteemipilte olema võimalik ka kustutada ning konteinereid pidi saama käivitada, peatada, kustutada ja uusi luua.

Töö raames valmis Angular.js raamistikul põhinev Cumulocity veebiliidese moodul, mis suhtles Cumulocity API-liidese, küsides Cumulocity andmelaost andmeid seadme Dockeri süsteemipiltide ja konteinerite kohta. Moodul pidi oskama neid kuvada ülejäänud veebiliidese sarnases stiilis. Samuti pidi moodul oskama saata Dockeri põhiseid operatsioone. Seadmele loodi agent, mis saatis iga 5 sekundi tagant andmeid Dockeri süsteemipiltide ja konteinerite kohta. Seade pidi oskama kuulata Cumulocity andmelattu saabuval operatsioonil ning vastavalt nendele käivitama Dockeri käsura programmi süsteemipiltide ja konteinerite haldamiseks.

Cumulocity platvorm on väga paindlik tööriist, mis võimaldab omavahel integreerida erinevaid tehnoloogaid, mis teevad asjade interneti seadmete haldamise lihtsaks. Kui Cumulocityga seotud seade on võimeline Dockeri konteinereid jooksutama, siis loodav agent võimaldab seadmel olevat tarkvara reaalajas hallata otse Cumulocity veebiliidese kaudu. Kõike seda saab teha läbi brauseri ja tänu Cumulocityle on võimalik hallata Dockeri konteinerite kaudu töötavat tarkvara mitmetes seadmetes samaaegselt.

Varasemalt oli töö autoril väikene kokkupuude Dockeriga ning selle konteinerite ja süsteemipiltidega. Cumulocity platvormiga kokkupuude oli esmakordne ja see oli töö juures kõige keerulisem osa.

Viidatud kirjandus

- [1] D. Crockford. The application/json media type for javascript object notation (json). <http://www.ietf.org/rfc/rfc4627.txt> [Vaadatud 14.05.2018].
- [2] Cumulocity. Administration. <https://www.cumulocity.com/guides/users-guide/administration/> [Vaadatud 13.05.2018].
- [3] Cumulocity. Cumulocity examples. <https://bitbucket.org/m2m/cumulocity-examples/src/default/java-agent/> [Vaadatud 13.05.2018].
- [4] Cumulocity. Cumulocity rest implementation. <http://cumulocity.com/guides/reference/rest-implementation/> [Vaadatud 13.05.2018].
- [5] Cumulocity. Cumulocity's domain model. <https://www.cumulocity.com/guides/concepts/domain-model/> [Vaadatud 13.05.2018].
- [6] Cumulocity. Interfacing devices. <http://cumulocity.com/guides/reference/rest-implementation/> [Vaadatud 13.05.2018].
- [7] Docker. Docker overview. <https://docs.docker.com/engine/docker-overview/> [Vaadatud 14.05.2018].
- [8] Tutorialspoint. Restful web services - introduction. https://www.tutorialspoint.com/restful/restful_introduction.htm [Vaadatud 14.05.2018].

II. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina, **Markus Peterson**,

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose

Konteinerite kaughaldus IoT seadmetes

mille juhendaja on Pelle Jakovits

- 1.1 reprodutseerimiseks säilitamise ja üldsusele kättesaadavaks tegemise eesmärgil, sealhulgas digitaalarhiivi DSpace-is lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
 - 1.2 üldsusele kättesaadavaks tegemiseks Tartu Ülikooli veebikeskkonna kaudu, sealhulgas digitaalarhiivi DSpace'i kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.
3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tartus, 14.05.2018