TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Markus Peterson

Dockeri konteinerite kaughaldus IoT seadmetes

Bakalaureusetöö (9 EAP)

Juhendaja: Pelle Jakovits

Tartu 2018

Dockeri 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 produktsioonis. 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 ja konteinerite paigaldamist.

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

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

Remote management of Docker 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 and container deployment.

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

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

Sisukord

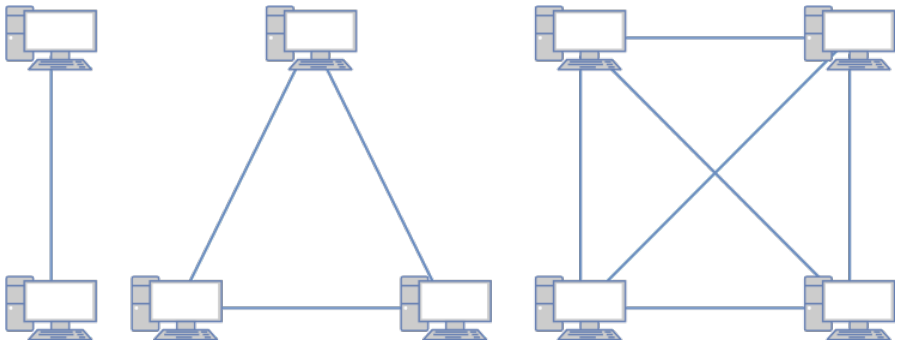
1 Sissejuhatus

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.

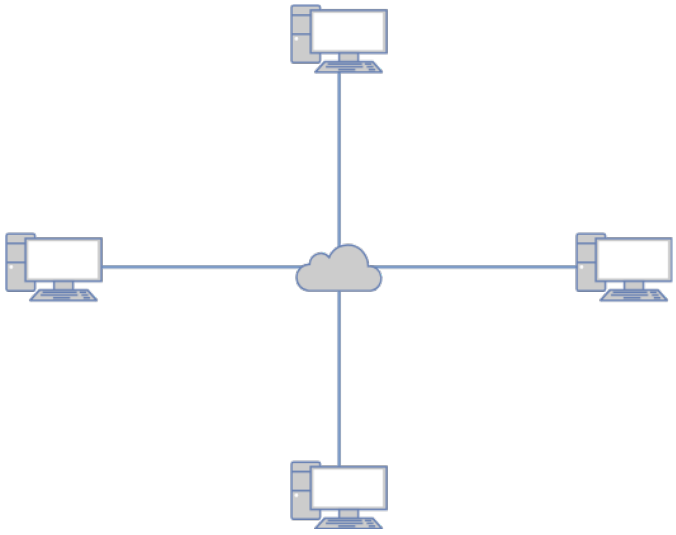
Matkaline jõuab külmal talvepäeval metsast koju ja soovib, et teda ootaks kodus ees kuum elektrisaun. Tal oli kogu retke jooksul käe peal nutikell, mis saab tema kehatemperatuuri jälgides aru, kui ta end läbi külmunult tunneb. Nutikell on samuti varustatud ka GPS seadmega ja tuvastab, kui kasutaja marsruudil koju on. Nende vaatluste järgi saadab nutikell kodus olevale eletrisaunale 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 tavapärases 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 tsenraliseeritud süsteemiks, mida illustreerib joonis 2.



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 [5]. Integreerides Dockeri Cumulocity platvormi, saab kontrollida seadmetel töötavaid rakendusi ning neid kergelt uuendada.

1.2 Lahendus

Pragusel hetkel ei ole leida lahendust, mis ühendaks omavahel Cumulocity platvormi ja Dockeri. 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 süsteemipiltidesse, mida saab lihtsasti liigutada ning käivitada seadmes, millesse on Docker installitud. Süsteemipildi käivitamisel tehakse sellest konteiner, milles

pakendatud tarkvara töötab isoleeritult [5]. Kasutaja peab saama näha platvormi veebikeskkonnas seadmetel olevaid süsteemipilte ja konteinereid. 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 veebiliidese moodul seadmete peal töötava Dockeri informatsiooni näitamiseks ning konteinerite 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 [8].

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 konteineritest ning kuidas neid läbi Dockeri käsurea hallata. Neljas peatükk kirjeldab nõudeid loodavale Cumulocity liidesele, arhitektuurist, kuidas liides toimib, ja liidese loomisest endast ning 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, konfigureerimist ja pakub reaalajalist logimist ning seadmete monitoorimist [11, 10]. 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 ja Kontron jaoks on loodud agendid, mida saab edasi arendada vastavalt vajadusele. Leidub ka Windowsile, Linuxile ja MacOSile mõeldud agente.

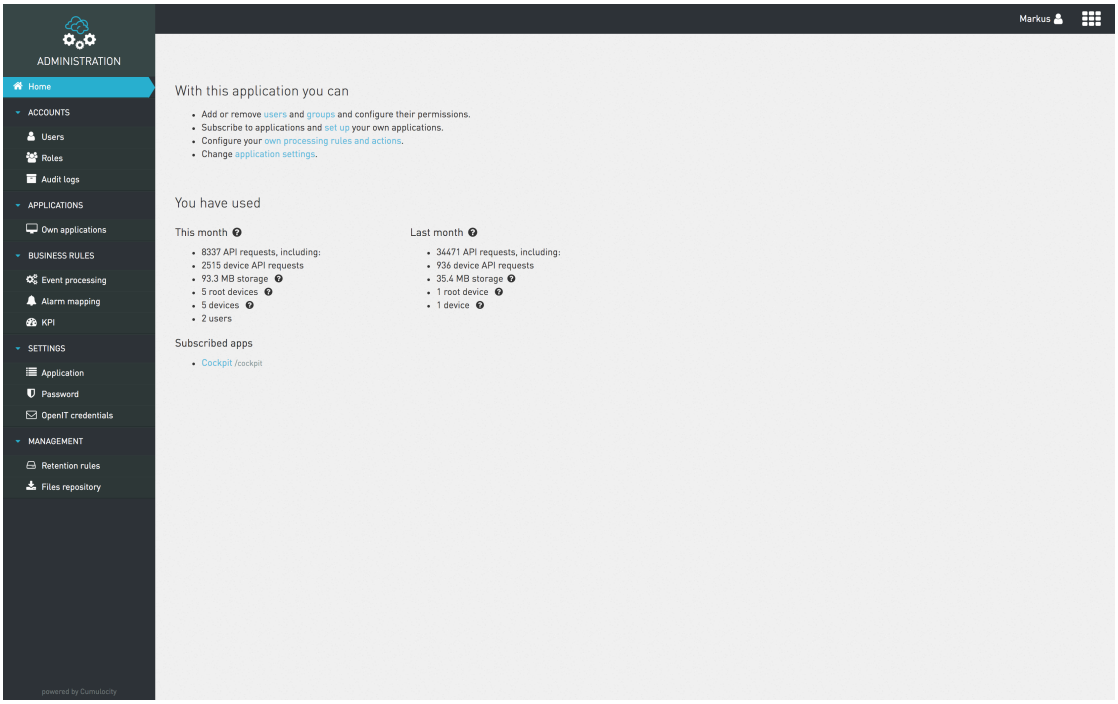
Cumulocity platvorm on üles ehitatud tenantite (eesti keeles üürnik) peale. Tenant kujutab endast kasutajate grupi üksust, läbi mille on võimalik platvormiga integreeruda. Sellised üksused on üksteisega seotud läbi puu struktuuri, mis tähendab, et igal tenantil on ülem, kellest see tenant tuletatud on, 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 [7]. 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.

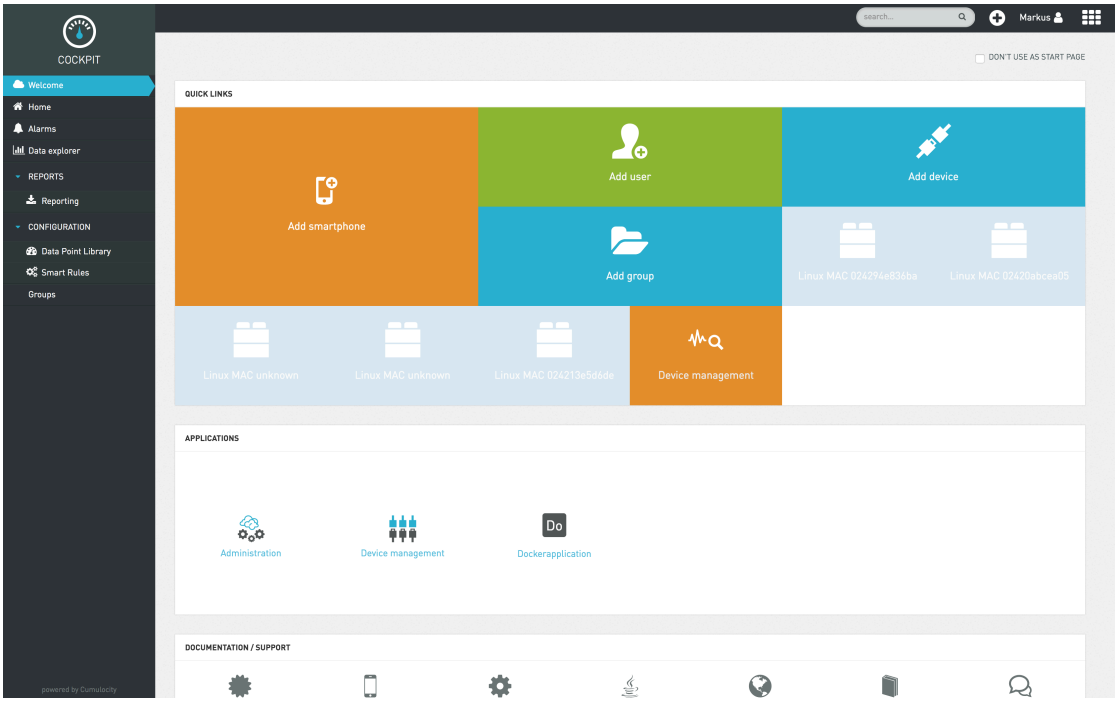
Administratsiooni rakenduse abil on võimalik lisada kasutajaid ja rolle ning neid

eemaldada; hallata tenanti rakendusi, nende seadistusi, konfiguratsioone ning andmete säilitamise reegleid; tenanti logi vaatamine; failide lisamine, alla laadmine 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.



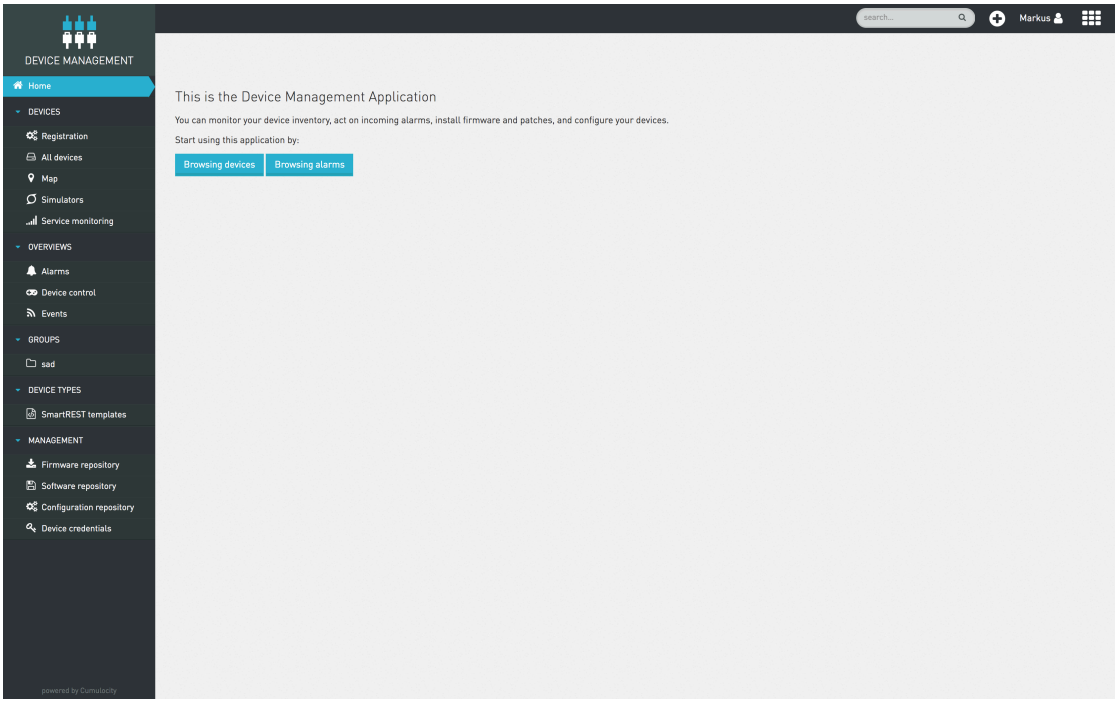
Joonis 3. Cumulocity administratsiooni rakendus

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 süsteemi operaator märgib alarmi lahendatuks. Operatsioonid tähistavad andmeid, mis on saadetud seadmele käivitamiseks või töötlemiseks [10]. Selleks võib olla näiteks teade elektrisaunale selle tööle panemiseks.

Seadmetel on võimalik kasutada andmelaoga suhtlemisel Cumulocity RESTful APIt (inglise keeles State Transfer Application Programming Interface). See on HTTP protokollil põhinev programm, mis järgib REST arhitektuuri stiile. 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 [9].

Tänapäeval on Javascripti populaarsuse tõttu tavaks kujunenud kasutada JSON formaati (inglise keeles JavaScript Object Notation), mis on kasutusel ka Cumulocity API-liideses. JSON formaat kasutab inimloetavat teksti, et edastada andmeobjekte, milleks on serialiseeritav väärtus näiteks sõne, number, tõeväärtus, järjend, attribuudi 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 curl[1] programmi abil GET päring vastu Cumulocity API-liidest. Päringu vastus on parema loetavuse mõttes formaaditud jq[2] 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 implementeerib tavaliselt 4 päringutüüpi [12]:

- 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 audentimisega (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 [9]. Joonisel 6 hoolitseb selle eest curl käsurea käsk, kuid allpool joonisel 7 on sellest pikem näide.

```
→ echo -n 'tenant/username:password' | base64
dGVuYW50L3VzZXJuYW1lOnBhc3N3b3Jk

→ curl -s -H "Authorization: Basic dGVuYW50L3VzZXJuYW1lOnBhc3N3b3Jk" "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. Audentimis 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 [9].

3 Dockeri tutvustus

3.1 Üldtutvustus

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

Dockeri eeliseks on lihtne rakenduse juurutamine, 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 süsteemipiltidega.

3.2 Dockeri süsteemipildid

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

Docker pakub võimalust hoida süsteemipilte DockerHub[3] keskkonnas. Sellest on võimalik süsteemipilte alla laadida, kasutades ”docker image pull” käsku. Süsteemipiltide kustutamiseks seadmelt on ”docker image rm” käsk.

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 container run”, siis konteineris on protsess isoleeritud - tal on oma failisüsteem, oma juurdepääsuvõrk ja oma peremeesarvutist eraldiolev isoleeritud protsessipuu. Olemasolevaid konteinereid

[3] https://hub.docker.com/

saab peatada ”docker container stop”, taas käivitada ”docker container start” või peatada ”docker container rm” käsu abil.

4 Praktiline osa

Käesolevas peatükis kirjeldatakse Cumulocity platvormi liidese loomist agendi ja veebiliidese mooduli näol. Kogu praktilises osas valminud kood on leitav autori Githubi repositooriumist (Lisa I. Lahenduse repositoorium).

4.1 Nõuded

Loodav Cumulocity liides jaguneb kaheks: veebiliidese moodul ja agent.

Veebiliideses seadmehaldus rakenduses seadme detailses vaates lisandub kaks vahekaarti:

1. Dockeri konteinerid
    1.1. Kasutaja peab nägema seadmes hetkel olevaid konteinereid
    1.2. Kasutajal peab olema võimalik eristada töötavaid konteinereid seisvatest
    1.3. Kasutaja peab saama saata seadmele operatsioone seisvate konteinerite käivitamiseks
    1.4. Kasutaja peab saama saata seadmele operatsioone töös olevate konteinerite peatamiseks
    1.5. Kasutaja peab saama saata seadmele operatsioone seisvate konteinerite kustutamiseks
    1.6. Kasutaja peab saama saata seadmele operatsioone uute konteinerite loomiseks seadmel olevatest süsteemipiltidest
        1.6.1. Konteinerite loomisel peab olema võimalus täpsustada konteineri argumente
        1.6.2. Konteinerite loomisel peab olema võimalus täpsustada konteineri sees jooksvat käsku
        1.6.3. Konteinerite loomisel peab olema võimalus täpsustada konteineri sees jooksva käsu argumente
2. Dockeri süsteemipildid
    2.1. Kasutaja peab nägema seadmes hetkel olevaid süsteemipilte
    2.2. Kasutaja peab saama saata seadmele operatsioone süsteemipiltide kustutamiseks
    2.3. Kasutaja peab saama saata seadmele operatsioone uute süsteemipiltide allalaadimiseks.

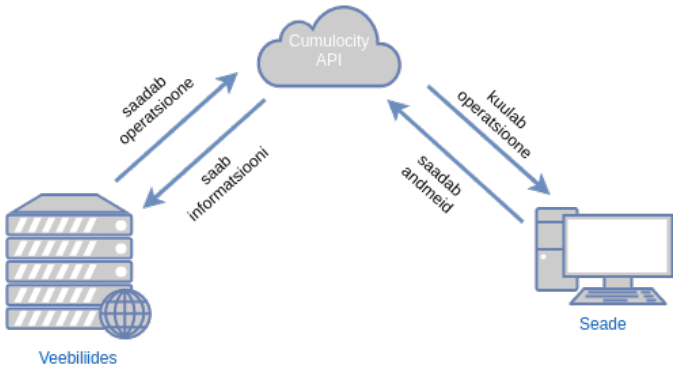
Cumulocity agendi nõuded

1. Agent peab oskama suhelda Dockeri käsurea programmiga
2. Agent peab saatma Cumulocity platvormile informatsiooni Dockeri süsteemipiltide ja konteinerite kohta
3. Agent peab oskama kuulata Cumulocity andmelattu saadetavaid operatsioone
4. Agent peab oskama Dockeri spetsiifilise operatsioonide puhul käivitada vastavat Dockeri käsurea programmi käsku
5. Agent peab olema lihtsasti seadmele integreeritav
6. Kasutaja peab saama seadet platvormiga integreerida seadme identifikaatori sisestamisega platvormi ning seadme aksepteerimisega platvormis.

4.2 Arhitektuur

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

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



Joonis 8. Veebiliidese ja agendi suhtlus Cumulocity API-liidesega

Veebiliidese moodul luuakse programmeerimiskeeles Javascript, kasutades Node.js käituskeskkonda. Node.js tuleb koos Npm paketihalduriga, 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, mida kasutatakse hüpertekst-märgistuskeeles (HTML) kirjutatud mallides, mille abil mooduli kasutajaliidese kuva üles ehitatakse. See tagab ühtse välimuse kogu Cumulocity veebikeskkonnas. 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 [8]. 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 agenti Dockeri funktsionaalsusega. Agendi sõltuvusi haldab Maven, mis on Java projekti- ja paketihaldussüsteem.

4.3 Cumulocity veebiliidese mooduli loomine

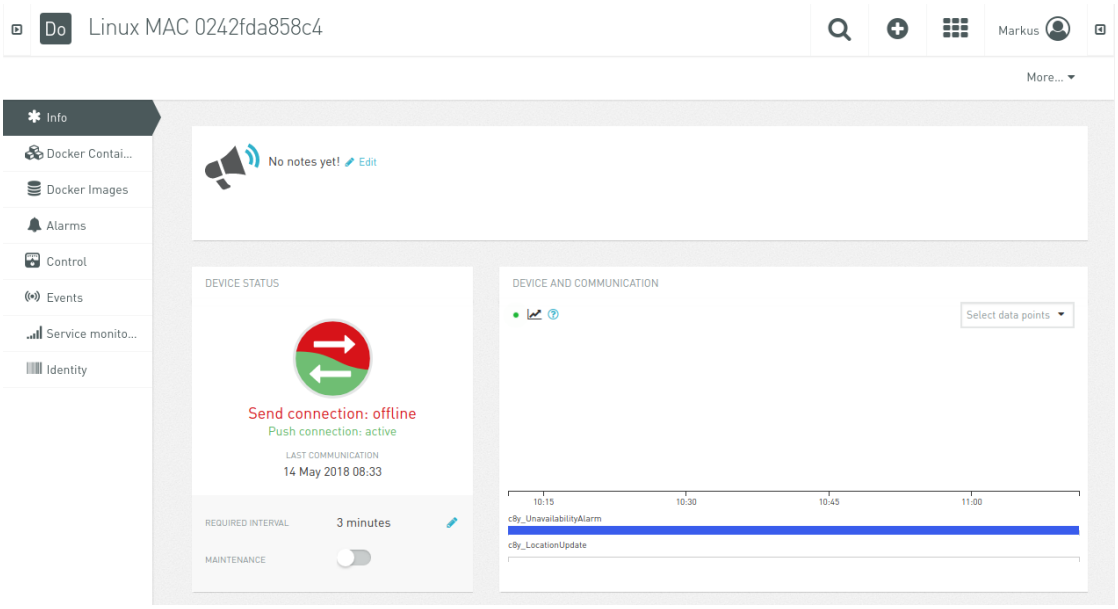
Veebiliidese moodul peab võimaldama kasutajal saada ülevaade seadmel olevate Dockeri konteinerite ja süsteemipiltide kohta. Süsteemipilte ja konteinereid peab saama kustutada ning konteinereid peab saama ka käivitada, peatada ja uusi luua.

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 <tenanti url>”, kus <tenanti_url> on Cumulocity tenanti 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.

```
(function () {
  'use strict';

  angular
    .module('myapp.dockerPlugin')
    .config(configure);

  /* @ngInject */
  function configure(c8yViewsProvider) {
    c8yViewsProvider.when('/device/:deviceId', {
      name: 'Docker Containers',
      icon: 'cubes',
      priority: 1000,
      templateUrl: ':::PLUGIN_PATH:::/dockerContainers/views/index.html',
      controller: 'dockerContainersCtrl'
    });

    c8yViewsProvider.when('/device/:deviceId', {
      name: 'Docker Images',
      icon: 'database',
      priority: 1000,
      templateUrl: ':::PLUGIN_PATH:::/dockerImages/views/index.html',
      controller: 'dockerImagesCtrl'
    });
  }

}());
```

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 abiklasse[4]. 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          }
```

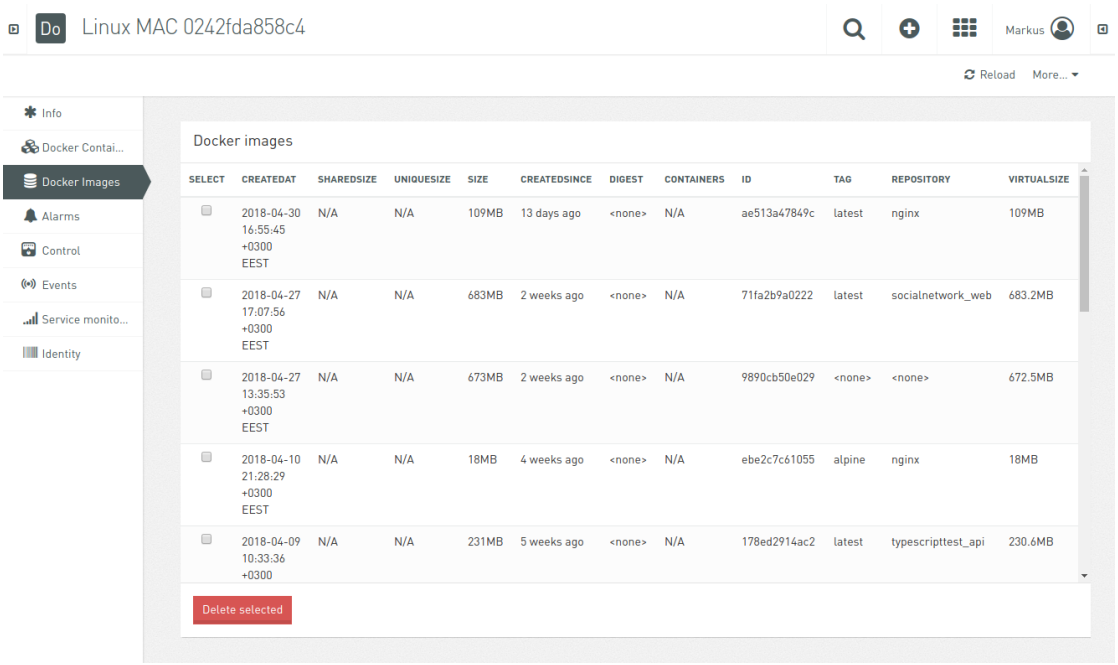
Joonis 11. Cumulocity API-st seadme Dockeri oleku päringine

Dockeri süsteemipiltide kuvamiseks kasutatakse hüpertekst-märgistuskeeles (HTML) kirjutatud malli, mida on näha joonisel 12. Selles on kasutatud Cumulocity UI poolt pakutavaid veebikomponente, mis tagab ühtse välimuse kogu ülejäänud platvormi veebiliidesega. 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

[4]http://resources.cumulocity.com/documentation/jssdk/latest//api/c8y.core

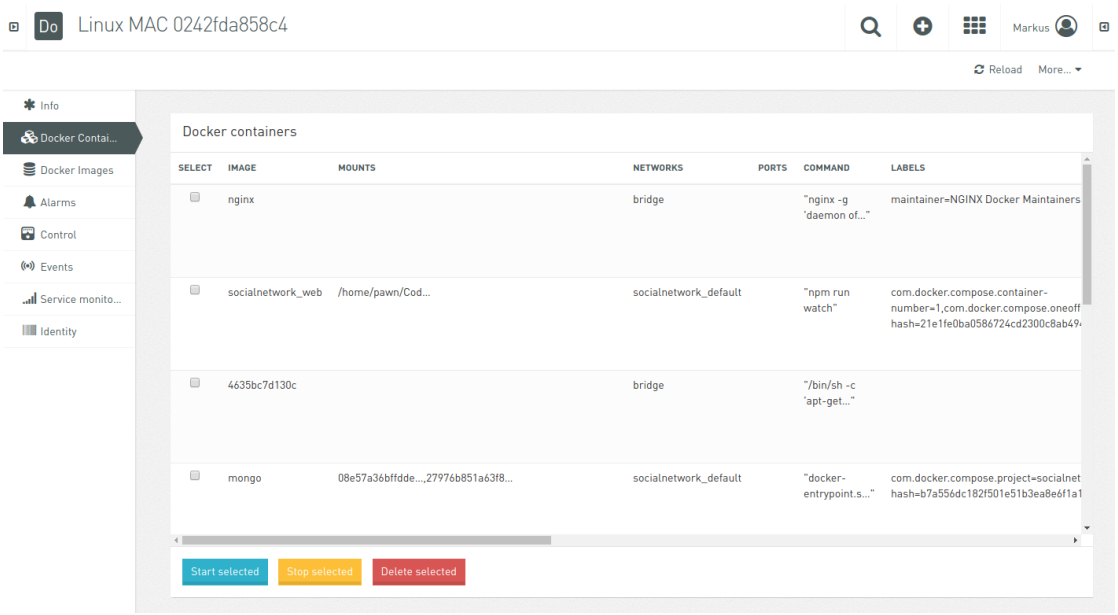


Joonis 13. Süsteemipiltide kuva Cumulocity seadmehaldus lehel

Dockeri 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.

```
<div class="card">
  <div class="card-header separator">
    <h4 class="card-title">Docker containers</h4>
  </div>
  <div style="max-height: 500px; overflow: auto;">
    <table class="table table-striped table-hover">
      <colgroup>
      </colgroup>
      <thead>
        <tr>
          <th>Select</th>
          <th ng-repeat="header in headers">{{ header }}</th>
        </tr>
      </thead>
      <tbody>
        <tr ng-repeat="container in containers" ng-class="{'success': container.is_active}">
          <td>
            <label style="display: block;">
              <input style="margin: 0 auto;" class="checkbox" type="checkbox" ng-click="selector.select(container.ID, $event.target.checked)"/>
            </label>
          </td>
          <td ng-repeat="key in headers">{{ container[key] }}</td>
        </tr>
      </tbody>
    </table>
  </div>
  <div class="card-footer separator">
    <button class="btn btn-primary" ng-click="startSelected()">
      Start selected
    </button>
    <button class="btn btn-warning" ng-click="stopSelected()">
      Stop selected
    </button>
    <button class="btn btn-danger" ng-click="deleteSelected()">
      Delete selected
    </button>
  </div>
</div>
```

Joonis 14. Mall konteinerite kuvamiseks

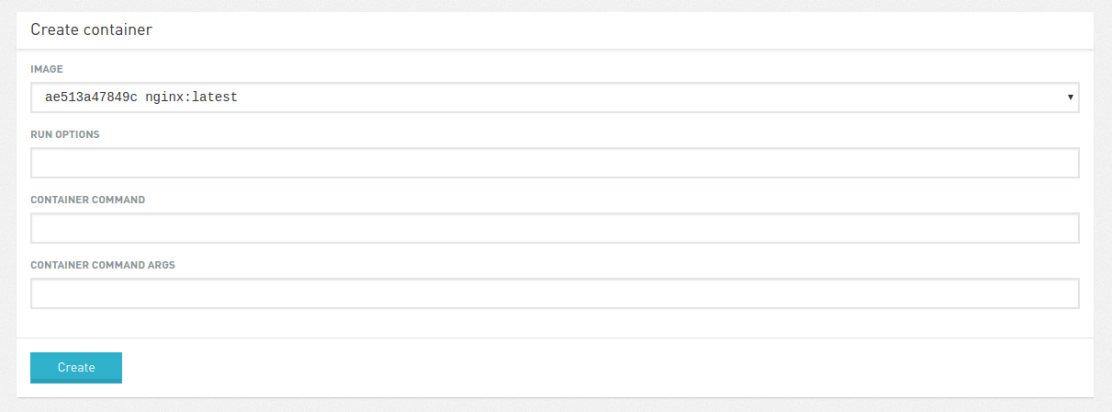


Joonis 15. Konteinerite kuva Cumulocity seadmehaldus lehel

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.

```
<div class="card">
  <div class="card-header-actions separator">
    <h4 class="card-title">Create container</h4>
  </div>
  <div>
    <div class="card-block">
      <div class="form-group">
        <label>Image</label>
        <select required class="form-control monospaced" ng-model="form.image">
          <option ng-repeat="image in images" value="{{ image.ID }}">
            {{ image.ID }} {{ image.repository }}:{{ image.tag }}
          </option>
        </select>
      </div>
      <div class="form-group">
        <label>Run options</label>
        <input class="form-control" type="text" ng-model="form.runOptions">
      </div>
      <div class="form-group">
        <label>Container command</label>
        <input class="form-control" type="text" ng-model="form.containerCommand">
      </div>
      <div class="form-group">
        <label>Container command args</label>
        <input class="form-control" type="text" ng-model="form.containerCommandArgs">
      </div>
    </div>
  </div>
  <div class="card-footer separator">
    <button class="btn btn-primary" ng-click="createContainer()">
      Create
    </button>
  </div>
</div>

<style>
  .monospaced {
    font-family: Consolas,Monaco,Lucida Console,Liberation Mono,DejaVu Sans Mono,Bitstream Vera Sans Mono,Courier New, monospace;
  }
</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 Dockeri mooduli nimeks ”c8y_Docker”. Et moodul teaks, mida operatsiooni kätte saamisel teha, määrame moodulile saadetavate argumentide sisse muutuja ”dockerCommand”. 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 - Süsteemipiltide kustutamine

- container rm - Konteinerite kustutamine
- container stop - Konteinerite peatamine
- container start - Kontainerite 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 süsteemipiltide 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 saata iga viie sekundi tagant Cumulocity andmelattu andmed Dockeri süsteemipiltide ja konteinerite kohta. Agent peab oskama kuulata operatsioonide sündmusi Cumulocity andmelaos ja seejärel vastavalt operatsioonile süsteemipilte või konteinereid kustutada või konteinereid käivitada, peatada, uusi luua.

Loodav agent põhineb Cumulocity näidisagendi koodil, millele luuakse ”docker-driver” nimeline moodul Dockeri spetsiifiliste funktsioonide jaoks. Sellest parema arusaama saamiseks soovitab autor korra vaadata Cumulocity näidisagendi projekti struktuuri [8]. Seadmel oleva Dockeriga suhtlemiseks kasutatakse Dockeri käsurea programmi. Selle välja kutsumisel peame arvestama, et see võib aega võtta ning seetõttu peab see olema asünkroone. Vastasel juhul jääb agent käsurea programmi lõpetamist ootama ning sellel ajal ei saa Cumulocity platvorm seadmega suhelda. Autor otsustas, et igale Dockeri spetsiifilisele Cumulocity operatsioonile peaks vastama klass, kus kutsutakse välja sellele operatsioonile vastav Dockeri käsurea programmi käsk. Selleks loodi CommandExecutor nimeline abstraktne klass, millest saavad operatsioonidele vastavad klassid pärineda.

```java
public abstract class CommandExecutor<T> implements Runnable {
    private Callback<T> callback = null;

    private static Logger logger = LoggerFactory.getLogger(CommandExecutor.class);

    BufferedReader systemCall(String ...command) throws IOException {
        ProcessBuilder pb = new ProcessBuilder(command);
        logger.debug("Executing command " + pb.command());
        Process process = pb.start();

        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;
    }

    public void setCallback(Callback<T> callback) {
        this.callback = callback;
    }

    @Override
    public void run() {
        T result = this.execute();

        if (callback != null) {
            callback.call(result);
        }
    }

    public abstract T execute();
}
```

Joonis 24. CommandExecutor klass, mille abil käsurea programme käivitatakse.

Joonisel 24 toodud CommandExecutor klass defineerib ”systemCall” meetodi, mis kasutab käsurea programmide käivitamiseks java.lang.ProcessBuilder klassi. See võtab sisendiks sõnede massiivi java.lang.String[] tüüpi objektina, mille näide on järgmine:

```
# Käsurea käsk
docker container run -p 80:80 nginx:alpine

# Sama käsk ProcessBuilderis
new String[] {"docker", "container", "run", "-p", "80:80", "nginx:alpine"}
```

Autor otsustas kasutada java.lang.Thread klassi CommandExecutor klassi objektide asünkroonseks käivitamiseks. Seetõttu on vajalik Runnable liidese ning sellega kaasne-

va ”run” meetodi implementeerimine. See kutsub välja ”execute” meetodi, mille peavad operatsioonidele vastavad alamklassid implementeerima.

Asünkroonsete tegevuste puhul ei ole teada, kuna lõppeb selle töö. Seetõttu on kasulik, et tegevus teavitaks mingit teist osa koodist enda lõpetamise puhul. Siin tuleb kasuks callback meetod, mis kutsutakse välja asünkroonse tegevuse lõpus. Callback meetod üldjuhul defineeritakse asünkroonse tegevuse loomisel, mistõttu on võimalik defineerida, mida käivitada tegevuse lõppemisel.

Ka CommandExecutor klassile on võimalik defineerida callback meetod. See on kasulik, kui käivitada CommandExecutor klass asünkroonselt java.lang.Thread klassi abil. Näiteks peale Dockeri käsurea programmi lõppemist saab callback meetodi sees saata Cumulocity platvormi uue informatsiooni Dockeri süsteemipiltide ja konteinerite kohta. Aga autor jättis võimaluse kutsuda CommandExecutor klassi ka sünkroonselt välja juhul kui on vaja koheselt tulemust teada saada. Sellisel juhul ei ole callback meetod vajalik. Et määrata callback meetodi tüüpi, loodi Callback nimeline liides, mis on nähtav jooniselt 25.

```
public interface Callback<T> {
    void call(T result);
}
```

Joonis 25. Callback liides, mis määrab callback meetodi tüübi.

Kasutades joonisel 24 näidatud CommandExecutor klassi, luuakse Dockeri spetsiifilistele Cumulocity operatsioonidele klassid, mis pärinevad CommandExecutor klassid. Kuna need erinevad põhiliselt ainult Dockeri käsurea programmi käsu poolest, siis autor toob välja ainult Dockeri süsteemipiltide kustutamiseks ja Dockeri konteinerite loomiseks loodud klassid, mis on nähtavad joonistel 26 ja 27.

```java
public class DockerImageRmCommand extends CommandExecutor {
    private OperationRepresentation operation;
    private String imageID;


    public DockerImageRmCommand(OperationRepresentation operation, String imageID) {
        this.operation = operation;
        this.imageID = imageID;
    }

    @Override
    public BufferedReader execute() {
        BufferedReader result;

        try {
            result = this.systemCall("docker", "image", "rm", this.imageID);
            this.operation.setStatus(OperationStatus.SUCCESSFUL.toString());
        } catch (IOException e) {
            result = null;
            e.printStackTrace();
            this.operation.setFailureReason(e.toString());
            this.operation.setStatus(OperationStatus.FAILED.toString());
        }
        return result;
    }
}
```

Joonis 26. Klass DockerImageRmCommand, mis võimaldab Dockeri süsteemipilte kustutada

```java
public class DockerContainerRunCommand extends CommandExecutor {
    private OperationRepresentation operation;
    private String imageName;
    private String containerOptions;
    private String containerCmd;
    private String containerCmdArgs;

    public DockerContainerRunCommand(OperationRepresentation operation,
                                     String imageName, String containerOptions,
                                     String containerCmd, String containerCmdArgs) {
        this.operation = operation;
        this.imageName = imageName;
        this.containerOptions = containerOptions;
        this.containerCmd = containerCmd;
        this.containerCmdArgs = containerCmdArgs;
    }

    @Override
    public BufferedReader execute() {
        BufferedReader result;
        try {
            ArrayList<String> command = new ArrayList<>();
            command.add("docker");
            command.add("container");
            command.add("run");

            if (this.containerOptions != null && !this.containerOptions.equals("")) {
                command.addAll(Arrays.asList(this.containerOptions.split(" ")));
            }

            command.add(this.imageName);

            if (this.containerCmd != null && !this.containerCmd.equals("")) {
                command.add(this.containerCmd);
            }
            if (this.containerCmdArgs != null && !this.containerCmdArgs.equals("")) {
                command.add(this.containerCmdArgs);
            }

            result = this.systemCall(command.toArray(new String[0]));
            this.operation.setStatus(OperationStatus.SUCCESSFUL.toString());
        } catch (IOException e) {
            result = null;
            e.printStackTrace();
            this.operation.setFailureReason(e.toString());
            this.operation.setStatus(OperationStatus.FAILED.toString());
        }
        return result;
    }
}
```

Joonis 27. Klass DockerContainerRunCommand, mis võimaldab Dockeri konteinereid luua

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 kuulab kõiki operatsioone andmelaos, siis peab moodul otsustama ka, kas operatsioon on mõeldud sellele seadmele või mitte.

Loodava agendi mooduli keskmeks on DockerDriver nimeline klass, mis suhtleb Cumulocity platvormiga. 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 joonisel 28 kirjeldatud kaks meetodit.

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

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

Joonis 28. Meetodid Dockerdriver klassis, 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 ”dockerCommand” 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.getProperty(supportedOperationType());
        String dockerCommand = (String) c8y_Docker.get("dockerCommand");

        CommandExecutor command = null;
        switch (dockerCommand) {
            case "image rm":
                command = new DockerImageRmCommand(operation, (String) c8y_Docker.get("imageID"));
                break;

            case "image pull":
                command = new DockerImagePullCommand(operation, (String) c8y_Docker.get("image"));
                break;

            case "container rm":
                command = new DockerContainerRmCommand(operation, (String) c8y_Docker.get("containerID"));
                break;

            case "container start":
                command = new DockerContainerStartCommand(operation, (String) c8y_Docker.get("containerID"));
                break;

            case "container stop":
                command = new DockerContainerStopCommand(operation, (String) c8y_Docker.get("containerID"));
                break;

            case "container run":
                command = new DockerContainerRunCommand(operation,
                        (String) c8y_Docker.get("imageName"),
                        "-d " + c8y_Docker.get("containerOptions"),
                        (String) c8y_Docker.get("containerCmd"),
                        (String) c8y_Docker.get("containerCmdArgs")
                );
                break;
        }
        if (command != null) {
            command.setCallback((Object result) -> this.updateDockerState());
            Thread thread = new Thread(command);
            thread.start();
        }
    }
}
```

Joonis 29. Meetod execute DockerDriver klassis, mis käivitab vastava Dockeri käsurea programmi sõltuvalt operatsioonist

Nüüdseks oskab DockerDriver klass võtta vastu veebiliidese moodulist saadetavaid Dockeri operatsioone. Selleks, et see oskaks saata informatsiooni Dockeri konteinerite ja süsteemipiltide kohta peab DockerDriverit laiendama. Selleks on c8y.lx.driver paketis olemas PollingDriver nimeline klass. See implementeerib java.lang.Runnable liidest, mistõttu on vaja DockerDriver klassis defineerida ”run” meetod. PollingDriver kutsub seda meetodit ajalise intervalli tagant välja, milleks on DockerDriver klassis määratud 5 sekundit. Selle meetodi sisse saab implementeerida platvormi informatsiooni saatmise, aga enne on vaja defineerida mudelid, milles hoida informatsiooni Dockeri konteinerite ja süsteemipiltide kohta. Nendeks on tavalised klassid, mis koos oma väljadega on informatsiooni esitusviisiks. Need on nähtavad joonistel 30 ja 31.

```java
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 30. Klass DockerContainer süsteemipiltide hoidmiseks

```
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 31. Klass DockerImage süsteemipiltide hoidmiseks

Informatsiooni Dockeri konteinerite ja süsteemipiltide kohta saab Dockeri käsurea programmist. See kuvab tavaliselt andmed tabelina, kus näiteks süsteemipiltide puhul kuvab üks rida ühe süsteemipildi andmed. Õnneks on Dockeri käsurea programmil olemas `--format '{{ json . }}'` lipp, mis kuvab andmeid JSON formaadis, kuid väljundiks on JSON objekti asemel read, kus üks rida on üks JSON objekt, mis tähistab ühte süsteemipilti. Seda on kerge parsida mudelitesse, milleks autor kasutas com.google.gson[5] teegist GSON klassi, kuna autor on sellega varasemalt kokku puutunud. Kuna Dockeri käsurea programmi väljundis on üks rida üks JSON objekt, siis tuleb parsida terve väljundi asemel igat rida eraldi. Dockeri käsurea programmi väljundi parsimist demonstreerib joonis 32.

```
Gson g = new Gson();
return reader.lines().map((s -> g.fromJson(s, DockerImage.class))).collect(Collectors.toList());
```

Joonis 32. Dockeri käsurea programmi väljundi parsimine DockerImage mudeliteks. Muutuja ”reader” on klassi BufferedReader objekt.

Seadmel olevate Dockeri süsteemipiltide ja konteinerite kättesaamiseks loodi joonisel 24 olevast CommandExecutor klassist joonistel 33 ja 33 klassid. Mõlemad käivitavad

[5]https://github.com/google/gson

Dockeri käsurea programmi, mis tagastab kummalegi andmed süsteemipiltide ja konteinerite kohta ning seejärel parsitakse need andmed joonisel 32 näidatud viisil mudeliteks.

```
public class DockerImageLsCommand extends CommandExecutor {
    public List<DockerImage> execute() {
        try {
            BufferedReader reader = this.systemCall("docker", "image", "ls", "--format", "{{ json . }}");

            Gson g = new Gson();
            return reader.lines().map((s -> g.fromJson(s, DockerImage.class))).collect(Collectors.toList());
        } catch (IOException e) {
            e.printStackTrace();
            return null;
        }
    }
}
```

Joonis 33. Klass DockerImageLsCommand, mis küsib Dockeri käsurea programmilt süsteemipiltide andmed ja parsib need DockerImage objektideks

```
public class DockerContainerLsCommand extends CommandExecutor {
    @Override
    public List<DockerContainer> execute() {
        try {
            BufferedReader reader = this.systemCall("docker", "container", "ls", "-a", "--format", "{{ json . }}");

            Gson g = new Gson();
            return reader.lines().map(s -> g.fromJson(s, DockerContainer.class)).collect(Collectors.toList());
        } catch (IOException e) {
            e.printStackTrace();
            return null;
        }
    }
}
```

Joonis 34. Klass DockerContainerLsCommand, mis küsib Dockeri käsurea programmilt süsteemipiltide andmed ja parsib need DockerContainer objektideks

Dockeri süsteemipiltide ja konteinerite Cumulocity platvormi saatmise lõpetamiseks on vaja veel defineerida DockerDriver klassi ”run” meetod, mida nõudis PollingDriver klass. Selles küsitakse informatsiooni Dockerilt ning see saadetakse Cumulocity andmelattu, mida on näha joonisel 36. Cumulocity andmelaos Dockeri süsteemipiltide ja konteinerite ühiseks hoiustamiseks loodi Docker nimeline klass, mille definitsioon on näha joonisel 35.

```
public class Docker extends HashMap<String, Object> {

}
```

Joonis 35. Docker klass Dockeri süsteemipiltide ja konteinerite ühiseks hoiustamiseks Cumulocity andmelaos

```
@Override
public void run() {
    updateDockerState();
}

private Docker getDockerState() {
    Docker dockerState = new Docker();
    dockerState.put("images", new DockerImageLsCommand().execute());
    dockerState.put("containers", new DockerContainerLsCommand().execute());
    return dockerState;
}

private void updateDockerState() {
    ManagedObjectRepresentation mo = new ManagedObjectRepresentation();
    mo.setId(gid);
    mo.set(getDockerState());
    inventory.update(mo);
}
```

Joonis 36. Meetodid, mille abil saadetakse infot Cumulocity andmelattu seadme Dockeri kohta

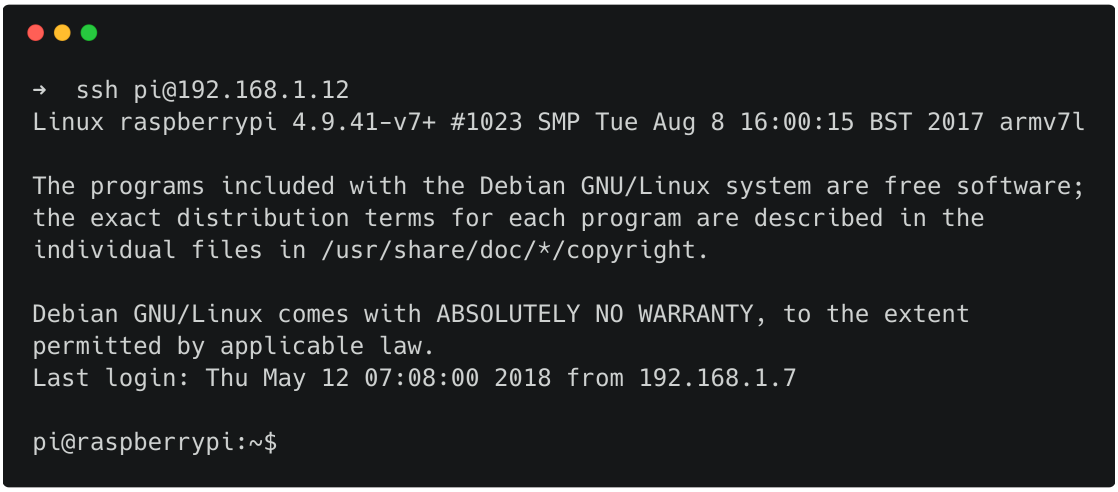
4.5 Lahenduse testimine

Lahenduse testimiseks kasutas autor värskelt seadistatud Raspberry Pi seadet, kuna autoril oli see seade olemas ning see sobitub hästi antud lahenduse kasutuslooga. Seadmele on varasemalt juba Dockeri installeeritud, kasutades Dockeri ametlikku installeerimisjuhendit [6]. Seadme integreerimiseks platvormiga kõigepealt kompileeriti agent kasutades näidisagendi Maveni ”package” käsku. See genereerib Raspberry Pi jaoks ”.deb” laiendiga paketi, mille abil saab agendi installida. Selleks tuleb see seadmele kopeerida. Autor kasutas selle jaoks scp nimelist käsurea programmi, mille abil saab faile kopeerida seadmetele üle võrgu [3]. Joonisel 37 on demonstreeritud scp programmi kasutamist. Kopeeritav fail ”cumulocity-rpi-agent_8.19.0_all.deb” asus arendusmasinas ”agent/java-agent/packages/rpi-agent/target/” kaustas ning see kopeeriti Raspberry Pi

seadmele, mis asus arendusmasinaga samas võrgus IP-aadressil ”192.168.1.12”. Seadmega ühendus tehti läbi ”pi” kasutaja ning kopeeritava faili lõppasukohaks oli selle kasutaja kodukaust Raspberry Pi seadmes(~).

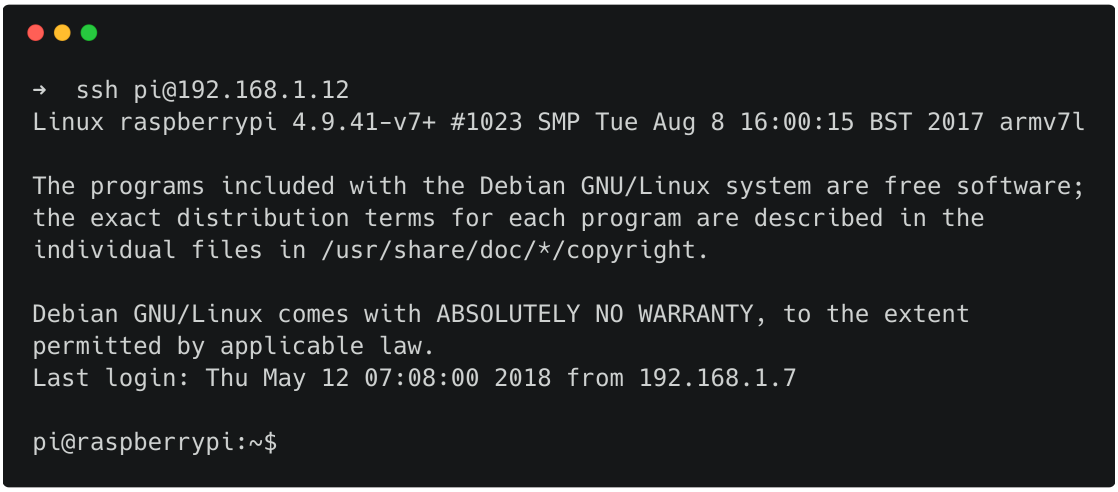
Joonis 37. Scp käsurea programmi kasutamine

Seejärel installeeriti see fail seadme peale. Selleks tuleb seadmesse sisse logida. Raspberry Pi puhul saab seda teha nagu tavalise arvutiga kasutades klaviatuuri, hiirt ja monitori. Autor otsustas selle asemel kasutada ssh ühendust, mis võimaldab saada seadme käsureale ligipääs üle võrgu [4]. Joonisel 38 on logitud arendusmasinaga samas võrgus IP-aadressil ”192.168.1.12” olevasse seadmesse ”pi” nimelisse kasutajasse. Sisse logimisel küsitakse ka selle kasutaja parooli, milleks Raspberry Pi seadmel on vaikimisi ”raspberry”.



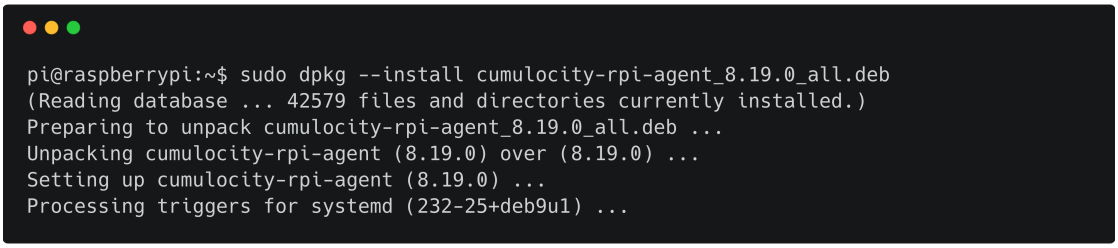
Joonis 38. Ssh käsurea programmi abil seadmesse logimine

Ssh ühenduse loomisel antakse ligipääs seadme käsureale, mis on nähtav viimasel real joonisel 38. Sealt on näha ka käsurea hetke aktiivne asukoht seadme failisüsteemis, mida markeerib peale koolonit olev ~, mis tähendab sisselogitud kasutaja kodukausta. See on sama kaust, kuhu joonisel 37 näidatud scp käsk installeerimispaketi kopeeris. Joonisel 39 käsurea programmi ls käivitamine näitab selle faili olemasolu praeguses aktiivses failisüsteemi asukohas.



Joonis 39. Käsurea programmiga ls praeguse aktiivse kausta sisu kuvamine.

Paketi installeerimiseks kasutati dpkg programmi koos ”–install” lipuga, mida kujutab joonis 40 [2]. Kui on soov enne paketi installimist veenduda, mis failid seadmele installitakse, siis saab kasutada ”–contents” lippu, mis tagastab failid ja nende installimisasukohad. Käsku jooksutati sudo käsu abil, kuna agent installitakse asukohta, mille sisu muutmiseks on vaja administraatori õiguseid.

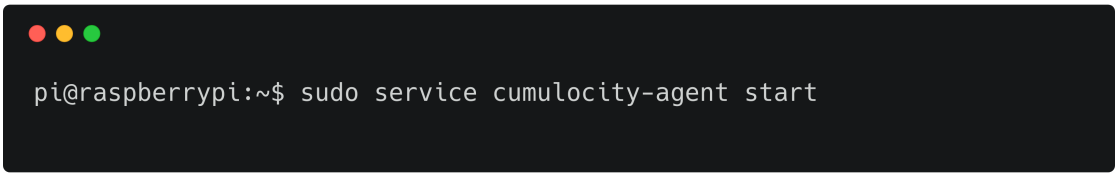


Joonis 40. Cumulocity agendi installimine dpkg programmi abil

Installi tulemusena kopeeriti paketi seest failid /usr/share/cumulocity-rpi-agent kausta. Selles Agent tuleb veel seadistada, et see teaks, millise tenantiga suhelda. Selleks on fail, mille asukohaks failisüsteemis on /etc/cumulocity-agent.properties. Faili muutmiseks kasutati nano käsurea programmi, mille abil kirjutati faili tenanti url ja nimi. Joonisel 41 on näha faili sisu peale muutmist.

Joonis 41. Näide Cumulocity agendi seadistusest

Seejärel käivitati agent. Agendi installeerimispakett installeris skripti, läbi mille on võimalik agenti käivitada kui teenust. Teenuste haldamiseks on service nimeline käsurea programm, mille abil on võimalik teenuseid käivitada, peatada, taaskäivitada ning nende olekut vaadata. Agendi käivitamist on näidatud joonisel 42. On võimalik, et installimise käigus agent juba käivitati, siis tuleb asendada joonisel 42 toodud käsus ”start” sõnaga ”restart”.

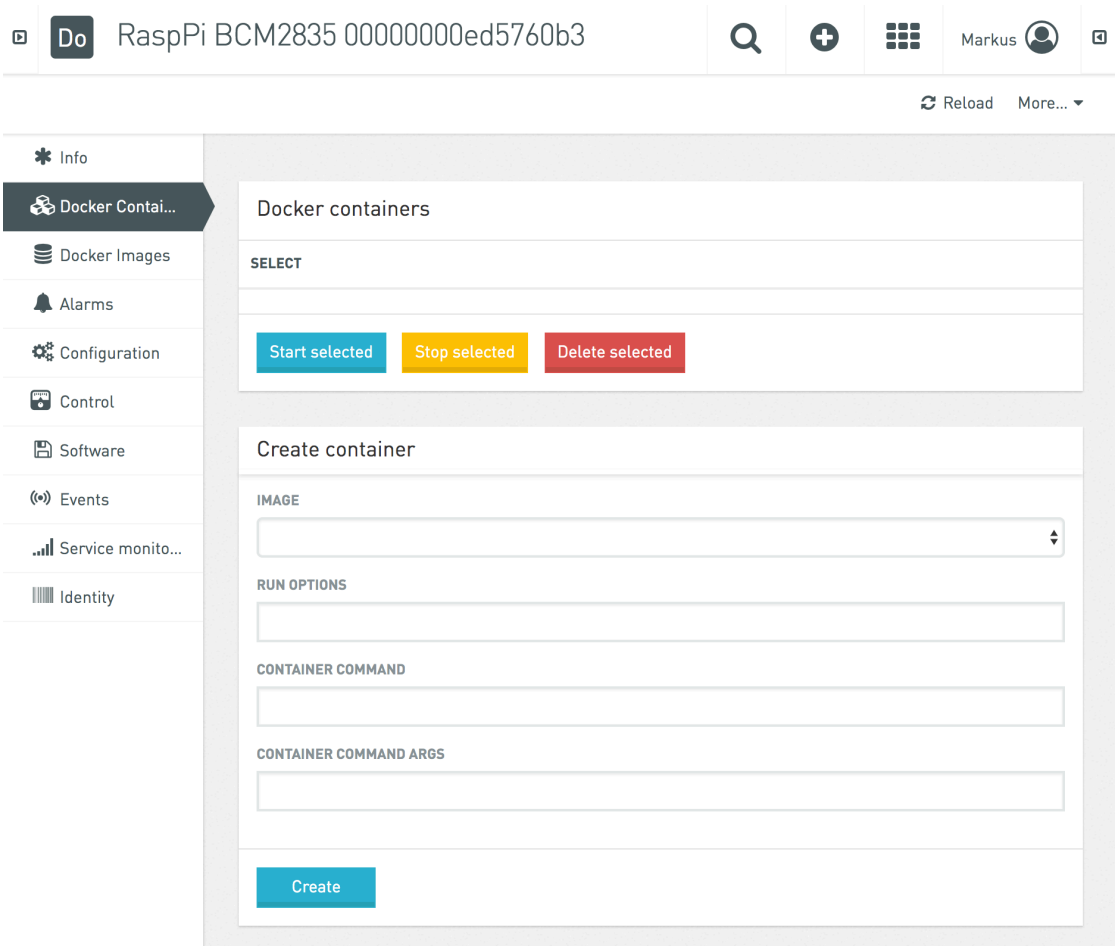


Joonis 42. Cumulocity agendi käivitamine teenusena

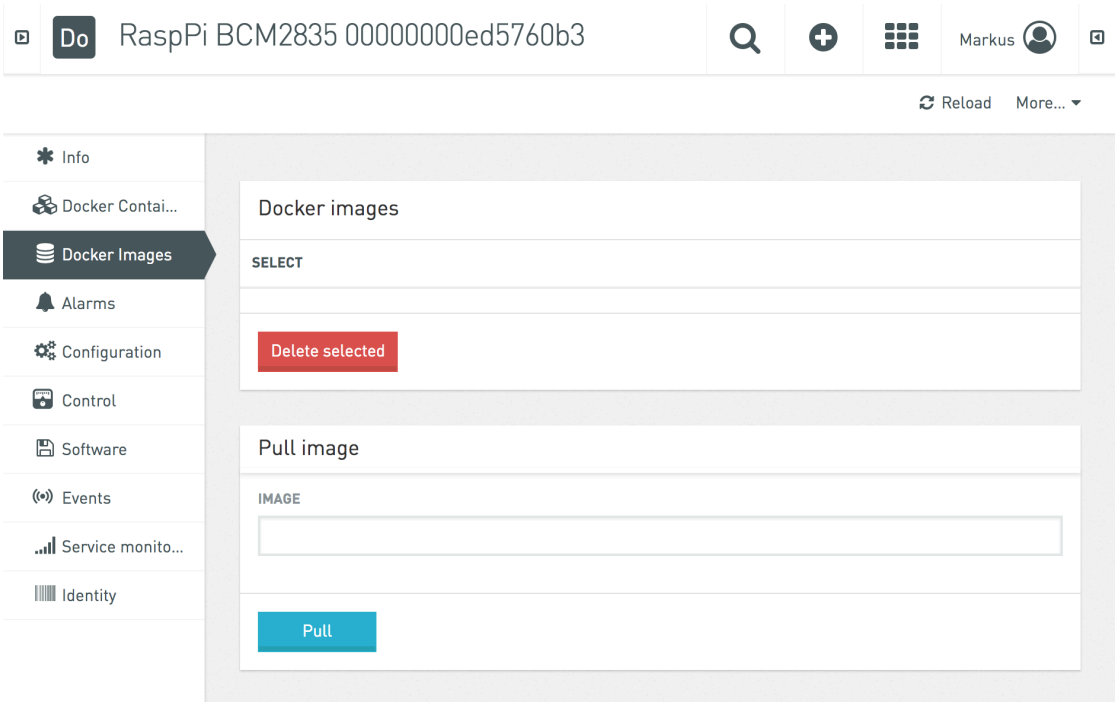
Käsu tulemuseks midagi ei väljasta. Seetõttu on kasulik vaadata agendi logi, mille järgi saab veenduda, et agent käivitus korralikult ning et mingeid tõrkeid ei tekkinud. Agendi logi salvestatakse süsteemi logisse, mis failisüsteemis asub /var/log/syslog asukohas.

Seadme registreerimise lõpetamiseks lisati Cumulocity veebikeskkonnas seadmehaldus rakenduse alt registreerimisvaatesse kirje uue seadme kohta. Kirje nõuab seadme ID väärtust, milleks seadme seerianumber. Raspberry Pi puhul on see nähtav /proc/cpuinfo failist, mille lõpus on kirje ”Serial”, millele järgneb numbritest ja tähtedest koosnev jada. See jada tuleb sisestati veebikeskkonda. Seepeale lubab Cumulocity seadmel platvormiga ühenduda ning seadmel olev agent saadab seadme kohta andmed. Seejärel kinnitati veebikeskkonnas seadme õigsust, millega seadme registreerimine lõppeb. Seade ilmub Cumulocity veebikeskkonnas seadmehaldus rakenduse all kõigi seadmete vaatesse.

Dockeri konteinerite ja süsteemipiltide vaadete tabelid on tühjad, kuna seadmes ei ole ühtegi süsteemipilti ega konteinerit veel (Joonis 44 ja 43).



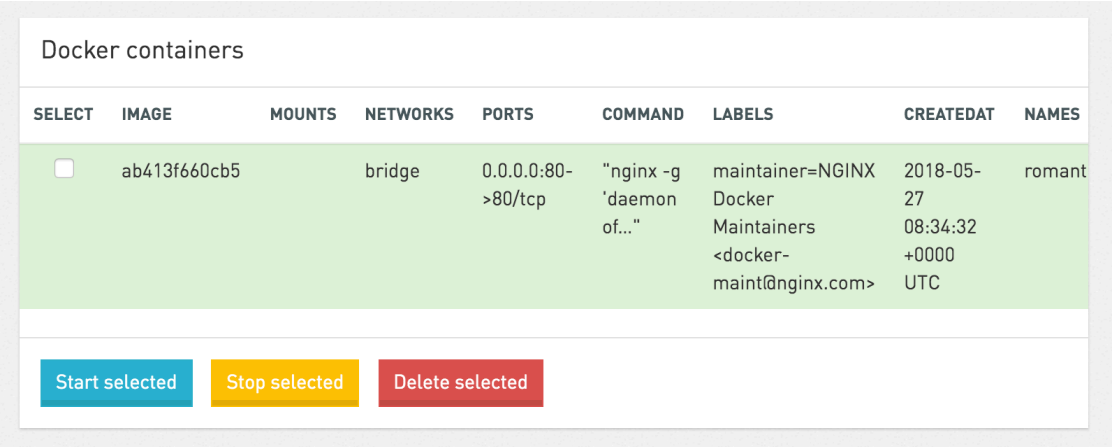
Joonis 43. Konteinerite vaade testimisseadmel



Joonis 44. Süsteemipiltide vaade testimisseadmel

Testimiseks kasutati nginx:latest nimelist süsteemipilti, mis sisaldab endas Nginx[6] veebiserveri tarkvara. See on vabalt kättesaadav ning kuna tegemist on veebiserveriga, mis töötab pordil 80, siis on võimalik selle töötamist seadmel valideerida curl käsu abil. Süsteemipildi allalaadimiseks kasutati süsteemipiltide vaates ”Pull image” alamsektsiooni, kuhu sisestati ”nginx:latest”. Kuna süsteemipilt tuleb alla laadida, siis sõltuvalt internetiühenduse kiirusest on allalaadimiseks kuluv aeg erinev. Siis uuendati vaadet kasutades ”reload” nuppu vaate paremal üleval nurgas, mida on näha jooniselt 44. Süsteemipilt nginx:latest ilmus süsteemipiltide vaates olevasse tabelisse.

Järgnevalt loodi sellest süsteemipildist konteiner, kasutades konteinerite vaates olevat ”Create container” alamsektsiooni. Süsteemipildiks valiti nginx:latest ning ”run options” väljale sisestati `--publish 80:80`, et siduda seadme port 80 konteineri pordiga 80. See tagab, et konteineris töötav veebiserver oleks kättesaadav väljaspool konteinerit. Seejärel vajutati ”Create” nuppu ja uuendati vaadet ”reload” nupuga. Tulemusena ilmus konteinerite vaates tabelisse aktiivne konteiner (Joonis 45).



Joonis 45. Aktiivne Dockeri konteiner töötamas Raspberry Pi seadmel

Konteineris jooksva veebiserveri ligipääsu testimiseks tehti curl käsurea programmi abil GET päring vastu seadme port 80, mis on nähtav jooniselt 46. Tulemusena tagastati tervitussõnum, mis tähendab, et konteineris veebiserver tõepoolest töötab ning see on seadme pordilt 80 kättesaadav.

[6]https://www.nginx.com/

```
→ curl 192.168.1.12:80
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Joonis 46. Raspberry Pi seadmele tehtud päring konteineris töötava veebiserveri valideerimiseks

Seejärel konteiner peatati, uuendati vaadet ning tabelis muutus äsja aktiivne olnud konteineri roheline tagataust valgeks. Sama konteiner uuesti käivitades muutus taust uuesti roheliseks. Konteineri kustutamiseks peatati konteiner uuesti ning seejärel kustutati konteiner. Tabelit uuendades kadus konteiner tabelist ja konteinerite tabeli seis on samasugune nagu joonisel 43. Seejärel kustutati nginx:latest süsteemipilt ja süsteemipiltide tabeli seisu tulemus on sama nagu joonisel 44.

Sellega on valideeritud uue tarkvara allalaadimine Dockeri süsteemipildina, sellest konteineri loomine koos konteinerile argumentide täpsustamisega, konteineri peatamine, käivitamine ja kustutamine ning süsteemipildi kustutamine.

4.6 Edasiarendused

Veebiliidese moodulis peab hetkel manuaalselt uuendama Dockeri konteinereid ja süsteemipilte tabelites. Kasutajamugavuse poolest oleks hea, kui need uuendused toimuksid reaalajas ning kasutaja ei peaks ”reload” nuppu vajutama peale igat operatsiooni. Samuti võiksid esineda abitekstid, mis tutvustaksid liidest esmakordsele kasutajale. Seadmele operatsioonide saatmise puhul ei uuenda agent automaatselt operatsioonide staatust ”successful” olekusse, kuna operatsioonide delegeerimine moodulitesse ning nende käivitamine toimub agendi poole pealt sünkroonselt, kuid operatsiooni sisene tegevus on asünkroone. Liideses on implementeeritud põhilised Dockeri funktsioonid, aga liidest on võimalik arenendada neid uusi funktsioone lisades. Näiteks lisada Dockeri volüümide või võrkude haldamine.

5 Kokkuvõte

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

Töö raames valmis Angular.js raamistikul põhinev Cumulocity veebiliidese moodul, mis suhtleb Cumulocity API-liidesega, küsides Cumulocity andmelaost andmeid seadme Dockeri süsteemipiltide ja konteinerite kohta. Moodul oskab neid kuvada ülejäänud veebiliidesega sarnases stiilis. Samuti võimaldab moodul saata Dockeri põhiseid operatsioone. Seadmele loodi agent, mis saadab iga 5 sekundi tagant andmeid Dockeri süsteemipiltide ja konteinerite kohta. Agent oskab kuulata Cumulocity andmelattu saabuvaid operatsioone ning vastavalt nendele käivitada Dockeri käsurea 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. Selle kasu on eriti näha asjade interneti seadmetes, kuna suurtes süsteemides muutub nende tarkvara haldamine keeruliseks. Loodud lahendus, mis ühendab omavahel Dockeri ja Cumulocity platvormi, on autori teades ainulaadne.

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] die.net. dpkg(1) - linux man page. https://linux.die.net/man/1/dpkg [Vaadatud 26.05.2018].

[3] die.net. scp(1) - linux man page. https://linux.die.net/man/1/scp [Vaadatud 26.05.2018].

[4] die.net. ssh(1) - linux man page. https://linux.die.net/man/1/ssh [Vaadatud 26.05.2018].

[5] Docker. Docker overview. https://docs.docker.com/engine/docker-overview/ [Vaadatud 14.05.2018].

[6] Docker. Get docker ce for debian. https://docs.docker.com/install/linux/docker-ce/debian/ [Vaadatud 26.05.2018].

[7] Cumulocity GmbH. Administration. https://www.cumulocity.com/guides/users-guide/administration/ [Vaadatud 13.05.2018].

[8] Cumulocity GmbH. Cumulocity examples. https://bitbucket.org/m2m/cumulocity-examples/src/default/java-agent/ [Vaadatud 13.05.2018].

[9] Cumulocity GmbH. Cumulocity rest implementation. http://cumulocity.com/guides/reference/rest-implementation/ [Vaadatud 13.05.2018].

[10] Cumulocity GmbH. Cumulocity's domain model. https://www.cumulocity.com/guides/concepts/domain-model/ [Vaadatud 13.05.2018].

[11] Cumulocity GmbH. Interfacing devices. http://cumulocity.com/guides/reference/rest-implementation/ [Vaadatud 13.05.2018].

[12] Tutorialspoint. Restful web services - introduction. https://www.tutorialspoint.com/restful/restful_introduction.htm [Vaadatud 14.05.2018].

Lisad

I. Lahenduse repositoorium

Autori poolt loodud lahendus on kättesaadav Githubi repositooriumist, mis asub aadressil `https://github.com/marcus17777/bachelor_thesis`. Kaustas ”agent” leidub Cumulocity näidisagendil põhinev agent, mis sisaldab endas docker-driver nimelist moodulit, mis vastutab Dockeri süsteemipiltide ja konteinerite halduse eest. Kaustas ”webplugin” asub Cumulocity veebiliidese moodul. Repositooriumist on kättesaadav ka see sama dokument ja LaTeX allikas, mille põhjal see dokument loodi.

II. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina, **Markus Peterson**,

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

   Dockeri 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 DSpacet'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, 27.05.2018