LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

LIETU INTERNETS KĀ PAKALPOJUMS

BAKALAURA DARBS

Autors: **Alberts Saulītis**
Studenta apliecības Nr.: as14081
Darba vadītājs: docents Dr. Dat. Leo Trukšāns

RĪGA 2018

ANOTĀCIJA

Darba mērķis ir izstrādāt lietu interneta platformas arhitektūru, kura nodrošina daudzlietotāju atbalstu, ir viegli uzstādāma no galalietotāju skatupunkta, un kuru ir iespējams piedāvāt potenciāliem lietotājiem kā pakalpojumu, parūpējoties par tās uzstādīšanu un uzturēšanu.

Darba gaitā ir veikts OpenHAB un Home Assistant lietu interneta platformu funkcionalitāšu apskats, salīdzinājums un to piemērotības izvērtējums daudzlietotāju arhitektūras izveidošanai.

Darba rezultātā ir izveidots Home Assistant platformas arhitektūras risinājums, kurš ļauj veikt vienkāršu sākotnējo platformas uzstādīšanu, nodrošina daudzlietotāju atbalstu, izmantojot vairākas platformas instances vienā resursdatorā, un parūpējas par lietotāju datu drošību.

Atslēgvārdi: lietu internets, OpenHAB, Home Assistant, MQTT, Arduino

ABSTRACT

INTERNET OF THINGS AS A SERVICE

The main goal of this paper is to create Internet of Things architecture which provides multiuser support, is easy to setup from end user perspective and which can be provided to potential clients as a service by taking care of platform setup and maintenance.

During paper, the Home Assistant and OpenHAB Internet of Things platform functionality has been mutually reviewed and compared. An evaluation has been created to find out which platform is the most suitable for multiuser environment.

As a result of the paper Home Assistant platform architecture solution which allows to perform easy initial setup of platform, supports multiuser environment using multiple instances of platform and takes care of user data security was created.

Keywords: Internet of Things, OpenHAB, Home Assistant, MQTT, Arduino.

Satura rādītājs

APZĪMĒJUMU SARAKSTS

ACL – Piekļuves vadības saraksts kurš norāda resursu piekļuves politiku.

AWS (Amazon Web Services) – mākoņpakalpojumu platforma, kura piedāvā mākoņskaitļošanas pakalpojumus.

Certbot – Rīks, kurš izmantojot Letsencrypt ļauj automatizēt SSL sertifikātu iegūšanu.

Docker – Atvērtā pirmkoda programmatūra, kura nodrošina operētājsistēmas līmeņa virtualizāciju, pazīstamu arī kā konteinerizāciju.

EC2 – AWS tīmekļa serviss, kurš nodrošina drošu, mērogojamu, mākoņskaitļošanas jaudu.

GP2 – AWS EC2 cietā diska tips

Letsencrypt – Uzticama sertifikātu autoritāte, kura piedāvā SSL sertifikātus par brīvu.

Mosquitto – Atvērtā pirmkoda programmatūra, kura nodrošina MQTT ziņojumapmaiņas protokolu.

MQTT – Publicēt – parakstīties ziņojumapmaiņas protokols, kurš darbojas virs TCP/IP sakaru protokola un tiek bieži izmantots lietu interneta risinājumos.

MySQL – Atvērtā pirmkoda relāciju datubāžu pārvaldes sistēma.

OpenHAB – Uz Java bāzēta mājas automatizācijas platforma, kura nodrošina daudzu ražotāju ierīču apvienošanu un pārvaldīšanu vienuviet.

OSGi – Java ietvars, kurš apraksta modulāras sistēmas un servisa platformu, kas implementē pilnīgu un dinamisku komponenšu modeli.

PaaS – Mākoņpakalpojumu kategorija, kurā tiek piedāvāts lietotājiem izmantot platformas bez sarežģītības, kas nepieciešama, lai izveidotu un uzturētu infrastruktūru.

RFID – Tehnoloģija, kas izmantojot radio frekvences ļauj identificēt ierīces.

Route 53 – AWS serviss, kurš nodrošina augstas pieejamības un mērogojamības domēnu nosaukumu sistēmas pakalpojumus.

RSS – Procesa izmantotā RAM atmiņas daļa.

S3 – AWS serviss kurš nodrošina augsti mērogojamu objektu uzglabāšanas servisu.

SCADA – Sistēma, kas sastāv no programmatūras un datortehnikas, kura ļauj kontrolēt industriālus procesus, uzraudzīt, uzkrāt un apstrādāt reālā laika datus, tieši mijiedarboties ar tādām ierīcēm kā sensoriem, motoriem un citām ierīcēm.

URL – Adrese, kas pārlūkprogrammā norāda, kur var atrast kādu konkrētu interneta resursu.

Vides mainīgie – Dinamiski uzstādīta vērtība, kas var ietekmēt procesa darbību un tiek nodota izsauktajam procesam.

YAML – Cilvēkiem viegli izlasāma datu serializācijas valoda, parasti izmantota iestatījumu datnēs.

IEVADS

Lietu internets ir strauji augoša nozare, kurā tiek nodrošināta iespēja, izmantojot internetu, veikt sensoru kontroli. Tiek paredzēts, ka līdz 2020. gadam lietu internets sastāvēs no 30 miljardiem ierīču. Eksistē daudzas lietu interneta platformas, tomēr liela daļa no tām ir pieejamas tikai komerciālām vajadzībām un ir sarežģīti uzstādāmas. Autors ir saskāries ar brīvi pieejamu atvērtā pirmkoda lietu interneta platformu, kura tika izmantota, lai iepazīstinātu ar lietu interneta pamatelementiem, tomēr platformas uzstādīšana un izmantošana nebija triviāla, platformas iespēja piedāvāt vairāklietotāju atbalstu bija limitēta.

Bakalaura darba mērķis ir izstrādāt lietu interneta platformas arhitektūru, kura nodrošina daudzlietotāju atbalstu, ir viegli uzstādāma no lietotāju skatupunkta, un kuru ir iespējams piedāvāt potenciāliem lietotājiem kā pakalpojumu, parūpējoties par tās uzstādīšanu un uzturēšanu. Darba mērķa sasniegšanai ir nepieciešams apskatīt un salīdzināt atvērtā pirmkoda lietu interneta platformu funkcionalitāšu klāstu, pievēršot uzmanību iespējām automatizēt sākotnējo platformas uzstādīšanu, nepieciešamajiem sistēmas resursiem un iespējām atbalstīt daudzlietotāju vidi. Lai varētu piedāvāt platformu kā pakalpojumu, ir nepieciešams izveidot arhitektūru, kas ļautu izvietot vairākas platformas instances uz viena resursdatora.

Lai veiktu brīvi pieejamo platformu salīdzināšanu, tiek veikta platformu dokumentācijas un pieejamās funkcionalitātes apzināšana un salīdzināšana, to rezultāti tiek izmantoti, lai izveidotu daudzlietotāju lietu interneta arhitektūru.

Darbs sastāv no 3 daļām. Pirmajā daļā ir apskatīta lietu interneta attīstība un iespējamie arhitektūras risinājumi. Otrajā daļā ir apskatītas un savstarpēji salīdzinātas OpenHAB un Home Assistant lietu interneta platformas. Trešajā daļā ir aprakstīta un izveidota lietu interneta platformas arhitektūra, kura nodrošina daudzlietotāju atbalstu un vienkāršu sākotnējo uzstādīšanu.

1. PAŠREIZĒJĀ SITUĀCIJA

Šajā daļā ir aprakstītas lietu interneta sastāvdaļas un attīstība.

1.1. Lietu interneta attīstība

Lietu internets piedāvā iespēju dažāda izmēra un pielietojuma ierīcēm veikt saziņu ar lietotnēm. Gan izmantotās ierīces, gan izmantotās lietotnes veic autonomus lēmumus un datu apmaiņu starp tām, tādejādi nodrošinot vienotu tīklu [1]. Galvenokārt lietu interneta attīstības vīzija var tikt iedalīta sekojošajos aptuvenos laika posmos:

- Mājas un personīgās lietošanas – 2010. gadā parādoties brīvi pieejamiem gudrajiem sensoriem un lietu starpkomunikācijas standartizēšanas procesa dēļ;
- Uzņēmumu lietošanai – 2015. gadā parādoties RFID mazumtirdzniecībā un uzlabojoties SCADA sistēmu pieejamībai;
- Komunālo pakalpojumu sfērā – 2020. gadā veicot kritiskās infrastruktūras uzraudzību un viedtīklu ieviešanu;
- Transporta sfērā – 2025. gadā ieviešot viedo satiksmes pārraudzību, automātiski vadītas mašīnas [2].

Tiek paredzēts, ka līdz 2020. gadam lietu internets sasniegs 50. miljardus savienoto ierīču [3].

1.2. Lietu interneta arhitektūra

Lietu interneta arhitektūras lielākoties iedalās divos tipos – centralizētās un decentralizētās. Lielākajā daļā gadījumu tiek izmantota centralizēta sistēma, kura piedāvā pakalpojumus gudrajām ierīcēm. Centralizētās sistēmas parasti nodrošina notikumu apstrādi, ziņojumu nosūtīšanu, datu uzglabāšanu un apskati. Dažkārt tiek izmantotas decentralizētas sistēmas, kuras nodrošina tādus pakalpojumus kā vienādranga (peer-to-peer) apziņošanu, datu uzglabāšanu u.c. [4]

Nākotnes lietu interneta arhitektūrā ir jānodrošina sekojošā funkcionalitāte:

- Enerģijas apzināšana – atkarībā no ierīces pieejamības un prasībām ierīcēm ir jāizmanto piemērots komunikācijas protokols un jānodrošina iespēja doties

gulēšanas režīmā, gadījumos, kad ierīce netiek izmantota, tādejādi samazinot enerģijas patēriņu.

- Resursu kontrole – izmantotajām ierīcēm ir jāatrodas vidē, kura ir attālināti kontrolējama, kā arī jānodrošina bojājumpiecietība.
- Servisa kvalitāte – dažādām ierīcēm, piemēram, temperatūras sensoram istabā un temperatūras sensoram industriālā boilerī ir jābūt iespējai nodrošināt atšķirīgu servisa kvalitāti. Lietām, kuras izmanto reālā laika datus, ir jādod augstāka prioritāte kā pārējām.
- Savstarpējā savietojamība – lietu internetam attīstoties, arvien vairāk ražotāju radīs lietu interneta ierīces, kurās izmantotie komunikācijas standarti var būtiski atšķirties.
- Traucējumu apzināšanās – palielinoties izmanto lietu klāstam, palielināsies arī savstarpējās komunikācijas traucējumi. Arhitektūrai ir jābūt veidotai, ņemot vērā traucējumus, lai varētu nodrošināt pēc iespējas lielāku darbspējas laiku.
- Drošība – izmantotajām lietu interneta ierīcēm ir jāspēj aizliegt neautorizētu piekļuvi, ņemot vērā sistēmas datoraparatūras ierobežojumus [5].

1.3. Lietu interneta arhitektūras sastāvdaļas

Cisco materiāli nosaka, ka lietu interneta arhitektūra kopumā var iedalīties līdz 5 slāņiem, kur katram slānim ir noteikta funkcionalitāte. Pirmkārt, veidojas ierīču slānis (device layer), kurā atrodas mikrokontrolieri un to sensori, kas veic datu iegūšanu. Ja nepieciešams, var izmantot vietējo vārteju, kura apvieno vairākus protokolus un var veikt minimālu datu analīzi. Protokolu analīze būtiski atvieglo gala sistēmas uzstādīšanas sarežģītību. Treškārt, interneta līmeni, dati tiek nogādāti no iepriekšējiem slāņiem uz mākoņplatformu. Ceturtkārt, mākoņplatformas slānis piedāvā iespēju analizēt iegūtos datus un serverus datu uzglabāšanai un analīzei. Un, visbeidzot, dati nokļūst lietotnē, kas piedāvā iespēju automatizēt lietu interneta ierīču darbību [6].

2. LIETU INTERNETA PLATFORMU APSKATS

Daļā tiek apskatītas Home Assistant un OpenHAB lietu interneta platformas un to atbilstība izvirzītajiem kritērijiem, pateicoties kuriem tiek veikta platformu savstarpēja salīdzināšana.

2.1. Izmantotie kritēriji

Lai veiktu platformu savstarpēju salīdzināšanu, ir jānostāda kritēriji, pēc kuriem platformas tiks vērtētas, un jānorāda to nozīme.

2.1. tabula

Platformas pārbaudēm izmantotā servera specifikācija

Resursa nosaukums	**Apraksts**
CPU	Intel Xeon CPU E5-2676 v3 @ 2.40GHz
Pieejamo kodolu skaits	1
RAM	DDR3 2GB
Operētājsistēma	Ubuntu Linux 16.04 LTS
Operētājsistēmas kodola versija	4.4.0-1052-aws
Cietas disks	20GB GP2 tipa

Platformas tiek uzstādītas, izmantojot Docker rīku un attiecīgajām oficiālajām Docker pakotnēm. Docker rīks ir palaists uz AWS EC2 platformas t2.small tipa servera, kuram ir pieejami tabulā 2.1 redzamie sistēmas resursi.

Lai izvairītos no platformu licencēšanas izmaksām un tiktu dota iespēja tās pielāgot atbilstoši nepieciešamībai, tiek apskatītas tikai platformas, kurām nav licenču maksas un kurām pirmkods ir brīvi pieejams.

2.1.1. Kopienas aktivitāte

Kopienas aktivitāte ļaus secināt par izvēlētās platformas nākotnes attīstību un pašreizējo stāvokli. Lai novērtētu kopienas aktivitāti, tiks veikta izvēlētās platformas repozitoriju atbalstītāju skaita apskate, un, ja pieejams, repozitoriju atzarojumu skaits, cilvēku skaits, kuri repozitoriju ir pievienojuši izlasei.

2.1.2. Platformas dokumentācija

Lai veiksmīgi veiktu platformas uzstādīšanu, ir nepieciešama pārraugāma un viegli izmantojama dokumentācija. Dokumentācija tiks vērtēta, veicot caur MQTT klientu saņemtas vienības attēlošanu platformas lietotāja saskarnē. Tiek pieņemts, ka MQTT serveris ir iepriekš uzstādīts, platformas sākotnējā uzstādīšana ir iepriekš veikta, izmantojot Docker rīku, un uz MQTT servera "test/SI7021_temperature/state" tēmu tiek pārraidīts ziņojums ar pēdējo temperatūras vienību.

2.1.3. Daudzlietotāju atbalsts

Daudzlietotāju atbalsts tiks vērtēts, ņemot vērā iespēju izmantot veidnes un MQTT atklāšanas mehānismus, lai automatizētu platformas uzstādīšanu tās lietotājiem. Pateicoties šim kritērijam, ir iespējams noteikt platformas daudzlietotāju ieviešanas sarežģītības pakāpi.

2.1.4. Izmantotie sistēmas resursi

Tā kā platformas tiek uzstādītas izmantojot Docker rīku, ir iespējams apskatīt tikai platformas CPU un RAM noslogojumu, neņemot vērā resursdatora sistēmas procesus. Izmantojot Zabbix serveri un dockbix-agent-xxl rīku, tiks apskatīts pēdējās 6 stundās vidējais, maksimālais un minimālais RSS atmiņas izmantojums, vidējais un maksimālais CPU noslogojums. Veicot testēšanu, tiek izmantoti 2.1.2 nodaļā uzstādītie platformas iestatījumi.

2.1.5. Drošība

PaaS neatņemama sastāvdaļa ir iespēja vienlaicīgi uzturēt vairākus lietotājus, nodalot tos vienu no otra. Apskatot drošību, tiks ņemta vērā sekojošā funkcionalitāte:

- Tīmekļa saskarnes autentifikācijas atbalsts
- MQTT autentifikācijas atbalsts

2.2. OpenHAB platforma

OpenHAB aizsākumi ir meklējami 2010. gadā, kad Kai Kreuzer aizsāka OpenHAB platformas izstrādi, pateicoties mājas automatizācijas hobijam un vēlmei izveidot ražotāja neatkarīgu un aparatūras agnostisku iespēju integrēt mājas automatizāciju un izklaidi vienuviet [7]. 2017. gada februārī tika laista klajā OpenHAB otrā versija, kura uzlaboja lietojamību, dodot iespēju veikt lielu daļu platformas uzstādīšanu no tīmekļa vietnes, samazinot nepieciešamību tieši rediģēt datnes uz OpenHAB servera. Lai saglabātu esošo papildinājumu, atbalstīto ierīču un protokolu klāstu, lielākoties visi pirmās OpenHAB versijas papildinājumi ir saderīgi ar OpenHAB otro versiju. Papildus iepriekš minētajam arī tika izlaista OpenHABian programmatūra, kura būtiski atvieglo OpenHAB uzstādīšanu uz maza izmēra datoriem kā Raspberry Pi.

Uz doto brīdi OpenHAB ir 267 papildinājumi, izmantojot kurus ir iespējams atbalstīt ierīces un ar tām saistītos protokolus [8].

OpenHAB sevi uzskata par sistēmu, kas papildina citas sistēmas. Tiek izmantota abstrakcijas pakāpe, kurā tiek pieņemts, ka apakšsistēmas tiek uzstādītas un iestatītas neatkarīgi no OpenHAB platformas [9].

OpenHAB pirmkods ir rakstīts Java valodā un galvenokārt bāzēts uz Eclipse SmartHome ietvara, kā arī izmanto Eclipse Equinox un Apache Karaf, lai veicinātu OSGi izpildlaika vidi [10]. Tīmekļa servera funkcionalitāti nodrošina Java bāzētais Jetty. Pateicoties tam, ka OpenHAB ir bāzēts uz Java valodas, to ir iespējams izpildīt uz daudzajām operētājsistēmām, kuras atbalsta Java virtuālās mašīnas (JVM) izpildi, tajā skaitā Windows, Linux un Mac OS X.

OpenHAB dokumentācija nosaka, ka platforma galvenokārt sastāv no sekojošajiem elementiem:

- *Things* – lietas, kuras var būt fiziski pievienotas sistēmai, tomēr tās nav obligāti fiziskas ierīces. Viens no piemēriem ir tīmekļa serviss vai jebkurš cits pārvaldāms informācijas avots;
- *Channels* – kanāli, lietu piedāvāti pakalpojumi, kurus var apskatīt lietas dokumentācijā;
- *Binding* – papildinājumi, kuri var tikt uzstādīti, lai padarītu lietas pieejamas un veiktu vienību sasaisti ar fiziskām ierīcēm. Papildinājumi abstrahē lietas, padarot tos vieglāk pieejamus platformai;

- *Items* – vienības, kuras apraksta funkcionalitāti un var tikt izmantotas lietotāju saskrnēs un automatizācijas loģikā. Vienības var saturēt statusu un saņemt komandas [11].

2.2.1. OpenHAB kopienas aktivitāte

OpenHAB izmanto GitHub repozitoriju sadarbības tīmekļa vietni, lai padarītu programmatūras izstrādes procesu caurredzamu un ļautu projekta atbalstītājiem izveidot problēmu aprakstus un iesūtītu pirmkoda uzlabojumus.

2.2. tabula

OpenHAB repozitoriju aktivitātes pārskats

Repozitorija URL	**Atzarojumu skaits**	**Atbalstītāju skaits**	**Izlases skaits**
https://github.com/openhab/openhab-distro	234	40	526
https://github.com/openhab/openhab-docs	168	92	83
https://github.com/openhab/openhab2-addons	1440	220	768
https://github.com/openhab/openhab1-addons	1708	381	3473
https://github.com/openhab/openhab-core	97	28	50

OpenHAB platforma kopumā izmanto 5 repozitorijas, kur katrai ir noteikts mērķis:

- Openhab-distro – satur platformas uzstādīšanas bāzi, Apache Karaf un Eclipse Equinox iestatījumus;
- Openhab-docs – platformas un tās papildinājumu dokumentāciju;
- Openhab2-addons – platformas otrās versijas papildinājumus;
- Openhab1-addons – platformas pirmās versijas papildinājumus;
- Openhab-core – platformas pamata ietvars, kas apraksta to, kādi resursi ir pieejami un kā tie tiek savstarpēji apstrādāti.

Ņemot vērā tabulā 2.2 minēto informāciju, var secināt, ka platformas atbalstītāji lielākoties pievērš uzmanību platformas papildinājumiem, nevis tās pamata ietvaram, kas var novest pie situācijas, kurā platformas pamata ietvars negūst pietiekamu atsaucību no kopienas un tādejādi platformā būtiskas lietojamības problēmas netiek risinātas. Kā viens no piemēriem problēmai var tikt minēts GitHub apraksts notikumu reģistrēšanas problēmai, kurā norādīts, ka platformas lietotājiem pārskatāmā veidā nav iespējams apskatīt ar papildinājumiem saistītas problēmas, piemēram, faktu, ka papildinājums nav uzstādīts. Šāda tipa informācija ir atspoguļota tikai žurnālfailos, kuri nav apskatāmi no platformas tīmekļa vietnes [12].

2.2.2. OpenHAB platformas dokumentācija

OpenHAB platformas dokumentācija ir apskatāma publiski pieejamā tīmekļa vietnē, un tās pirmkods ir pieejams dokumentācijas repozitorijā.

Lai uzstādītu MQTT vienības parādīšanu, tika sekots iesācēja dokumentācijai, kurā ir aprakstītas izmantotās lietotāja saskarnes, papildinājumu uzstādīšana, vienkāršas vienības uzstādīšana un tās attēlošana, izmantojot vietnes karti. Katram no soļiem ir pievienots ekrānuzņēmums, kurš būtiski palīdz navigēt.

Sekojot papildinājumu uzstādīšanas pamācībai, ir parādīts, kā uzstādīt "Network binding", kurš ļauj atpazīt tīklā pieejamās ierīces un uztur informāciju par ierīču darbspējas laiku, tomēr, tā kā uzdevums ir atspoguļot MQTT vienību, tiek izvēlēts "MQTT binding" un tas veiksmīgi uzstādīts, kas tiek atspoguļots lietotāja saskarnē. Papildus tam, uzspiežot uz "MQTT binding" tiek atvērta attiecīgā papildinājuma dokumentācija, kurā ir aprakstītas iestatījumu datņu atrašanās vietas, kā arī klienta un vienību iestatījumu paraugi [13].

Vadoties pēc MQTT papildinājuma dokumentācijas, var secināt, ka, lai pilnīgi uzstādītu MQTT papildinājumu, ir jāveic servisa un vienības uzstādīšana. Sekojot dokumentācijā norādītajam paraugam, tiek iestatīts MQTT serviss. MQTT servisa iestatījumi ir apskatāmi 1. pielikumā, un tas tiek iestatīts "services/mqtt.cfg" datnē.

Pēc servisa iestatījuma, sekojot MQTT papildinājuma dokumentācijai, tiek veikta vienības uzstādīšana. Dokumentācijas paraugos ir dots temperatūras vienības atspoguļošanai nepieciešamais formāts, kuru var apskatīt attēlā 2.1.

```
Number temperature "temp [%.1f]"
{mqtt="<[publicweatherservice:london-
city/temperature:state:default]"}
```

Att. 2.1. **OpenHAB MQTT papildinājuma dokumentācijā atspoguļotais vienības paraugs**

Piemērojot doto paraugu izvēlētajiem kritērija priekšnosacījumiem, tiek iegūti attēlā 2.2 parādītie iestatījumi.

```
Number temperature "temp [%.1f]"
{mqtt="<[mqtt:test/SI7021_temperature/state:state:defau
lt]"}
```

Att. 2.2. **OpenHAB temperatūras vienības iestatījumi**

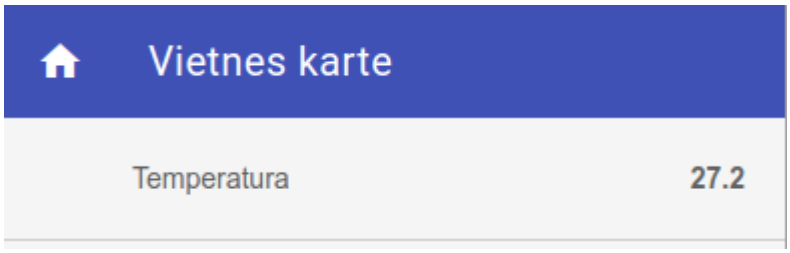
Diemžēl MQTT papildinājuma dokumentācijā nav atspoguļots, kādā veidā ir iespējams pārliecināties, ka vienība ir uzstādīta pareizi. Šī iemesla dēļ tika pieņemts, ka vienība ir uzstādīta pareizi un uzsākta vienības attēlošanas iestatīšana, sekojot iesācēja dokumentācijai.

Iesācēja dokumentācija norāda, ka, lai vienība tiktu parādīta, ir nepieciešams izveidot vietnes karti (sitemap), kura sevī ietver attēlojamās vienības. Vadoties pēc dokumentācijas, tika izveidota jauna vietnes karte, kura iekļauj iepriekš izveidoto temperatūras vienību. Izveidoto vietnes karti var apskatīt attēlā 2.3.

```
sitemap default label="Vietnes karte"
{
    Text item=temperature label="Temperatura"
}
```

Att. 2.3. **Izveidotā OpenHAB vietnes karte**

Lai vietnes karte tiktu atspoguļota tīmekļa vietnē, BasicUI lietotāja saskarnes iestatījumos ir jānorāda noklusētās vietnes kartes nosaukums. Pēc nosaukuma norādīšanas un, atverot BasicUI lietotāja saskarni, ir iespējams apskatīt izveidoto temperatūras vienību. Izveidotās vienības atspoguļojums ir redzams attēlā 2.4.



Att. 2.4. **Temperatūras vienības atspoguļojums OpenHAB BasicUI lietotāja saskarnē**

OpenHAB iesācēja un MQTT papildinājuma dokumentācija vidēji labi apraksta uzstādīšanas procesu, tomēr tajā netiek atspoguļotas iespējamās problēmsituācijas un iestatījumu pārbaudes iespējas, kā rezultātā problēmsituāciju risināšana var būt apgrūtināta.

2.2.3. OpenHAB daudzlietotāju atbalsts

OpenHAB papildinājums MQTT EventBus atbalsta automātisku MQTT pierakstīšanos tēmām, tomēr tā galvenais mērķis ir nodrošināt iespēju nodot citai OpenHAB platformām MQTT saņemtās tēmas saturu [13]. Papildus iepriekš minētajam, vienībai ir iepriekš jābūt definētai platformas iestatījumu datnēs, kas pilnībā izslēdz iespēju automātiski atklāt jaunas vienības un tādejādi papildinājums nepilda kritērijos nostādīto mērķi. MQTT EventBus papildinājuma paredzētās lietošanas UML diagramma ir apskatāma 2. pielikumā.

OpenHAB ļauj izmantot vides mainīgos, lai norādītu datņu atrašanās vietas, HTTP savienojumu klausīšanās portu un līdzīgu funkcionalitāti, tomēr tas neatbalsta iestatījumu uzstādīšanu no vides mainīgajiem, piemēram, nav iespējams uzstādīt MQTT servera atrašanās adresi, izmantojot vides mainīgos, kas būtiski apgrūtina daudzlietotāju atbalstu, jo visi iestatījumi ir obligāti jānorāda iestatījumu datnēs, un tos nav iespējams nodot OpenHAB lietotnei citā veidā.

Tā kā nav iespējams izmantot MQTT atklāšanas mehānismus un OpenHAB nav iebūvēta funkcionalitāte, kas atbalstītu iestatījumos izmantot vides mainīgos, ir nepieciešams izmantot resursdatora sistēmas funkcionalitāti un apstrādāt iestatījumu datnes pirms OpenHAB palaišanas. Viens no iespējamajiem veidiem, kā tas var tikt darīts, ir, izmantojot *envsubst* Linux utilītprogrammu, kura veic datņu apstrādi un aizvieto sastaptās vides mainīgo atsauces uz vides mainīgajos definētajām vērtībām.

OpenHAB platforma neatbalsta populārus veidņu izveides mehānismus, kā rezultātā ir jāizmanto ārpus platformas pieejama funkcionalitāte, kas būtiski apgrūtina platformas daudzlietotāju atbalsta nodrošināšanu.

Uzstādot OpenHAB platformu, eksistē trīs papildinājumu uzstādīšanas iespējas:

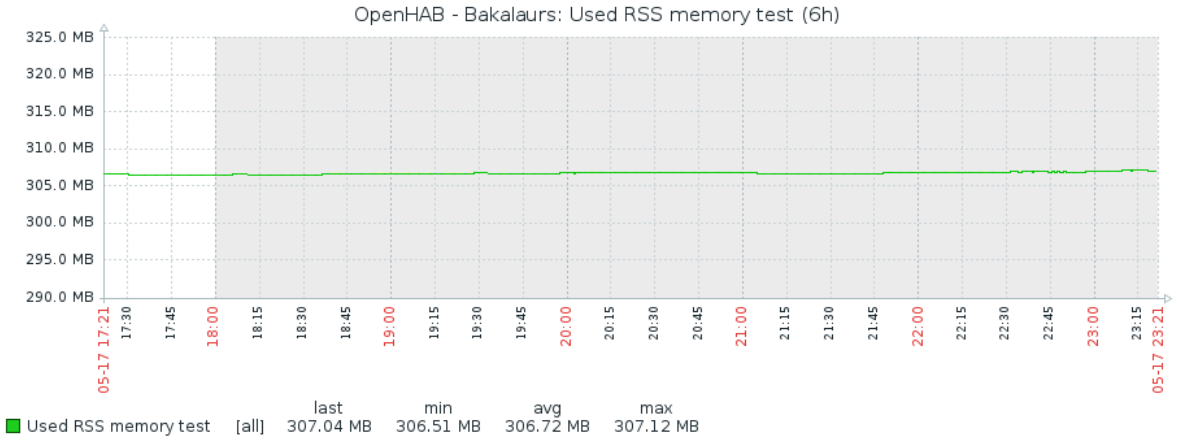
1. Veikt papildinājumu definīciju *addons.config* datnē
2. Veikt papildinājumu uzstādīšanu izmantojot tīmekļa vietni
3. Veikt papildinājumu uzstādīšanu izmantojot Karaf komandrindas saskarni

Veicot uzstādāmo papildinājumu definīciju *addons.config* datnē, tiek būtiski atvieglots uzstādīšanas process, jo datnē definētie papildinājumi tiek automātiski uzstādīti, palaižot OpernHAB platformu, tomēr tam arī ir negatīvs efekts – lietotājs nevar veikt papildinājumu uzstādīšanu, izmantojot 2. un 3. iespēju, jo papildinājumi, kas nav definēti *addons.config* datnē, tiek automātiski izdzēsti, pārlādējot OpenHAB platformu. Tā kā lietotājiem pievienot jaunus papildinājumus ir būtiska platformas sastāvdaļa, ir nepieciešams automatizēt papildinājumu uzstādīšanu, izmantojot Karaf komandrindas saskarnes *feature:install* komandu. Karaf

komandrindas saskarnes ir pieejama, izmantojot SSH protokolu, kuru nodrošina OpenHAB platforma.

2.2.4. OpenHAB izmantotie sistēmas resursi

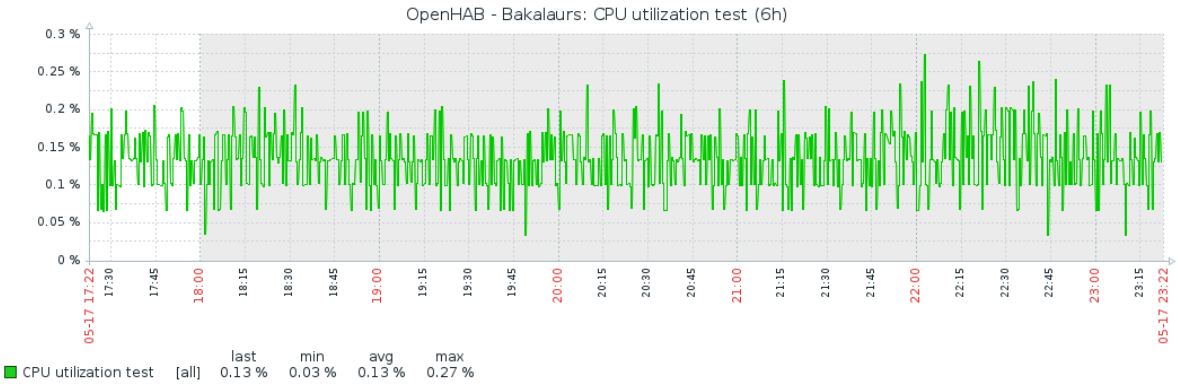
Veicot platformas RAM RSS izmantošanas apsekošanu 6 stundu garumā, izmantojot Zabbix rīku, tika izveidots attēlā 2.5 redzamais grafiks.



Att. 2.5. **OpenHAB RAM RSS atmiņas noslogojums 6 stundu periodā**

Informācija grafikā neliecina par tendenci palielināt izmantotās atmiņas apjomu pār laika periodu, tomēr izmantotās atmiņas apjoms ir augsts, ļoti iespējams, JVM izmantošanas dēļ.

Attēlā 2.6 ir redzams CPU noslogojums 6 stundu periodā, kas ir iegūts tādā pašā veidā, kā RAM RSS grafiks.



Att. 2.6. **OpenHAB CPU noslogojums 6 stundu periodā**

Grafiks neliecina par tendencēm palielināt CPU noslogojumu pār laika periodu un vidējā CPU noslodzes vērtība ir ļoti zema**.**

2.2.5. OpenHAB drošība

OpenHAB neatbalsta funkcionalitātes ierobežošanu noteiktam lietotājam, kā arī nepiedāvā nekādu iebūvētu funkcionalitāti, lai ierobežotu pieeju OpenHAB platformai, piemēram, izmantojot HTTP lietotāja autentifikāciju. Kaut arī dokumentācija norāda, ka OpenHAB platforma nevar implementēt HTTP autentifikāciju, ir norādīts, ka OpenHAB izmantotais tīmekļa serveris Jetty nesagādā problēmas, ja autentifikācija ir uzstādīta ar starpniekservera, piemēram, Nginx palīdzību [14].

OpenHAB MQTT papildinājums pilnībā piedāvā iespēju izmantot lietotāja un paroles autentifikāciju, norādot to iestatījumu datnēs, un atbalsta daļēju MQTT sertifikātu autentifikāciju. Lai varētu veikt MQTT sertifikātu autentifikāciju, ir nepieciešams veikt jaunas Java KeyStore datnes izveidi un pievienot gan klienta, gan sertifikātu autoritātes informāciju datnei. Keystore datnes atrašanās vietu ir iespējams nodot OpenHAB platformai, uzsākot darbu, izmantojot EXTRA_JAVA_OPTS vides mainīgo. Lai OpenHAB izmantotu sertifikātu autentifikāciju, MQTT servera pieslēgšanās parametrs servisa iestatījumos ir jāizmaina no TCP uz SSL [15].

Platforma ir iepriekš pētīta un ir secināts, ka platformas papildinājumiem netiek veikta integritātes validācija, un papildinājumu pirmkods netiek pienācīgi pārbaudīts pret drošības ievainojamībām. Papildus tam eksistē 9 OpenHAB papildinājumi, kuri var apskatīt visus notikumu maģistrāles ierakstus. Vairāk kā 100 papildinājumi neveic lietotāju ievades validāciju, tādejādi ļaujot potenciālam lietotājam, piekļūt pie resursiem ārpus platformas [16].

2.3. Home Assistant platforma

Home Assistant ir atvērtā pirmkoda platforma un tam ir Apache 2.0 licence. Platformas izstrādi aizsāka Paulus Schoutsen 2013. gada 17. septembrī [17]. Home Assistant kopumā ir pieejami 1070 papildinājumi [18]. Lielākā daļa iestatījumu ir pieejami iestatījumu datņu veidā, kuras ir formatētas YAML formātā [19]. Platforma ir bāzēta uz Python 3. versijas programmēšanas valodas, tādejādi samazinot slieksni, kas nepieciešams, lai uzsāktu platformas izstrādi. Lai nodrošinātu pirmkoda paredzēto funkcionalitāti, kopumā 94% no platformas pirmkoda aptver testi [20].

Home Assistant kodols atbild par mājas IoT ierīču kontroli, un tā kodols sastāv no 4 daļām:

- Event Bus – notikumu maģistrāle, kas klausās notikumus un reaģē uz tiem;

- State Machine – stāvokļu grafs, kurš uztur lietu stāvokli un izsauc state_changed mehānismu stāvokļu maiņas gadījumā;
- Service Registry – pakalpojumu reģistrs, kurš ļauj izsaukt servisus un reģistrēt jaunus servisus;
- Timer – taimeris, kurš reizi sekundē sūta uz notikumu maģistrāli jaunu notikumu [21].

Kodola savstarpējās komunikācijas grafisku attēlojumu var redzēt 3. pielikumā.

2.3.1. Home Assistant kopienas aktivitāte

Līdzīgi kā OpenHAB, Home Assistant arī izmanto GitHub repozitoriju sadarbības tīmekļa vietni un kopumā izmanto 3 repozitorijus pamatfunkcionalitātes nodrošināšanai.

2.3. tabula

Home Assistant repozitoriju aktivitātes pārskats

Repozitorija URL	Atzarojumu skaits	Atbalstītāju skaits	Izlases skaits
https://github.com/home-assistant/home-assistant	3243	983	14298
https://github.com/home-assistant/home-assistant-polymer	339	124	259
https://github.com/home-assistant/home-assistant.github.io	1976	1451	351

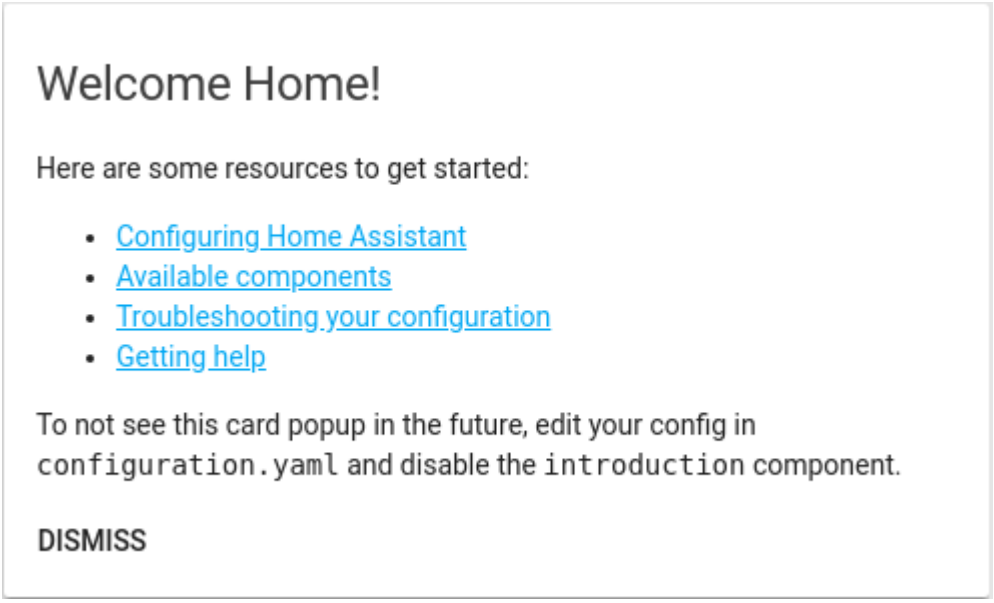
Katrai no 2.3 tabulā minētajām repozitorijām ir sekojošie mērķi:

- home-assistant – satur Home Assistant kodolu un iebūvētos papildinājumus;
- home-assistant-polymer – satur tīmekļa vietnes saskarni;
- home-assistant.github.io – satur Home Assistant platformas dokumentāciju.

Ņemot vērā 2.3 tabulā minētos datus, var secināt, ka Home Assistant ir būtisks kopienas ieguldījums, papildus tam Home Assistant dokumentācijai ir vairāk atbalstītāju nekā platformas kodolam, kas var liecināt par faktu, ka ne tikai pirmkoda autori ir ieinteresēti uzlabot dokumentāciju.

2.3.2. Home Assistant platformas dokumentācija

Home Assistant dokumentācija ir publiski pieejama tīmekļa vietnē, un tās saturs tiek kontrolēts, izmantojot home-assistant.github.io repozitoriju. Pirmajās lietošanas reizēs tiek parādīts palīdzības logs, apskatāms attēlā 2.7, kurš norāda uz dokumentāciju, lai uzsāktu pilnvērtīgi lietot Home Assistant.



Att. 2.7. **Home Assistant sākotnējās uzstādīšanas dialogs**

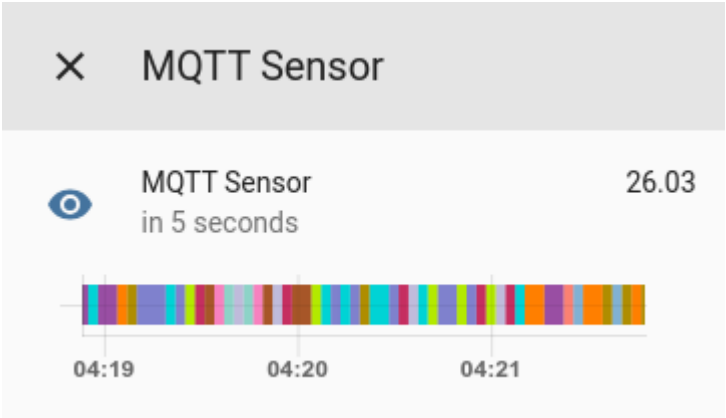
Tā kā Home Assistant lielākā daļa funkcionalitāts ir iebūvēta iekšā, nav nepieciešams veikt papildinājumu uzstādīšanu un var doties uz MQTT dokumentāciju. MQTT dokumentācija norāda, ka ir iespējams izmantot gan iebūvētu MQTT serveri, gan arī pieslēgties eksistējošam [22]. Iestatījumi tiek norādīti YAML formātā configuration.yaml datnē, dokumentācijā ir sīki aprakstīta katra uzstādīšanas nianse un tas, kā veikt iestatījumu testēšanu [23]. Diemžēl iestatījumu pārlāde nav atrunāta MQTT dokumentācijā, bet, apskatot dziļāk platformas tīmekļa vietni, iestatījumos ir dota iespēja pārbaudīt un pārlādēt iestatījumus. Izmantojot testēšanas pamācību, ir iespējams, izmantojot izstrādātāja iestatījumus veikt MQTT statusa nosūtīšanu, papildus tam, pamācībā ir aprakstīts arī, kā izmantot operētajsistēmas rīkus, lai pārbaudīta ziņojuma veiksmīgu saņemšanu.

MQTT pārbaudīšanas dokumentācijas izvēlnē ir pieejama iespēja doties uz sensoru uzstādīšanu, kur ir aprakstīts ar vairākiem paraugiem, kā saņemt vienkārši MQTT vērtību, pakāpeniski pieaugot izmantoto iestatījumu daudzumam, kas būtiski palīdz izprast kopējo piedāvāto funkcionalitātes klāstu. Vienkāršākais variants apraksta MQTT temperatūras vērtības saņemšanu, kas atbilst uzstādītajiem kritērijiem. Izmantotos temperatūras sensora iestatījumus var apskatīt attēlā 2.8.

```
sensor:
  - platform: mqtt
    state_topic: "test/SI7021_temperature/state"
```

Att. 2.8. **Home Assistant MQTT temperatūras sensora iestatījumi**

Pēc uzstādīšanas veikšanas tika pārlādēti iestatījumi un MQTT sensors tika atspoguļots tīmekļa saskarnē, kā redzams attēlā 2.9.



Att. 2.9 **Sensora atspoguļojums tīmekļa saskarnē**

Home Assistant dokumentācija ļauj vienkārši pievienot jaunus MQTT sensorus un pārbaudīt to darbību, izmantojot izstrādātāja rīkus tīmekļa vietnē. Papildus tam var redzēt, ka daļa no nepieciešamās funkcionalitātes. lai nodrošinātu IoT platformu, ir jau iepriekš implementēta, piemēram, vēsturisku datu saglabāšana, kas izmanto SQLite datubāzi [24].

2.3.3. Home Assistant daudzlietotāju atbalsts

Home Assistant atbalsta MQTT ierīču atklāšanu, tādejādi ļaujot veikt tikai minimālas iestatījumu izmaiņas iestatījumu datnēs – tikai lai ieslēgtu ierīču atklāšanu. Lai varētu veikt ierīču atklāšanu, mikrokontrolierim vai vārtejai ir jāveic Home Assistant iestatījumu nosūtīšana, izmantojot MQTT [25]. Home Assistant dokumentācija apraksta vairākus paraugus un norāda formātu, kurš ir jāizmanto, lai veiksmīgi notiktu iestatījumu atklāšana, izmantojot MQTT. MQTT tēmas formāts kopumā sastāv no 4 vai 5 apakštēmām, kuras norāda:

- <discovery_prefix>/ – prefikss, kuru izmanto Home Assistant, lai atklātu jaunas ierīces. Definējams iestatījumu datnēs;
- <component>/ – komponentes identifikators, piemēram, sensor, binary_sensor u.c;
- [<node_id>/] – neobligāts identifikators, kurš ļauj ierīcēm klausīties tikai savus MQTT ziņojumus;

- <object_id>/ – ierīces vai sensora identifikators;
- <> – tēma, kura apraksta pašreizējo darbību, piemēram, config vai state.

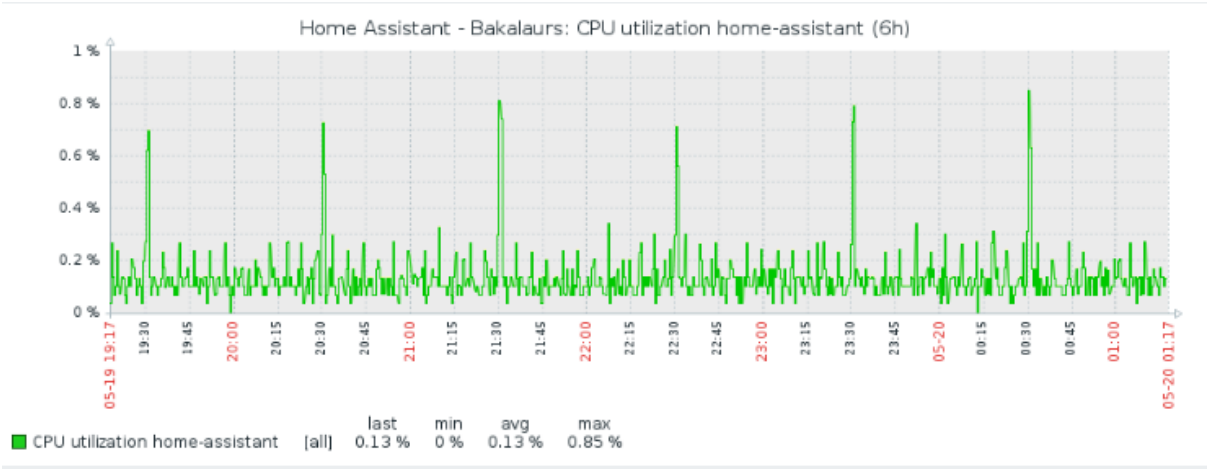
Apskatot MQTT tēmas formātus, var secināt, ka prefikss nevar tikt atdalīts ar lietotāja identifikatoru, piemēram, test/homeassistant/, tādejādi neļaujot izmantot MQTT iebūvētus pieejas kontroles mehānismus vairāku lietotāju arhitektūrā.

Home Assistant YAML iestatījumi ļauj izmantot vides mainīgo atsauces iestatījumu datnēs, tādejādi atvieglojot lietotāja specifisku iestatījumu izveidi, piemēram, MQTT servera adrese var tikt automātiski aizpildīta, izmantojot vides mainīgos [19].

Lai atvieglotu platformas lietojamību, ir iespējams izmantot papildinājumu, kurš ļauj labot iestatījumu datnes [26]. Papildinājums integrējas ar Home Assistant, izmantojot panel_iframe iebūvēto papildinājumu.

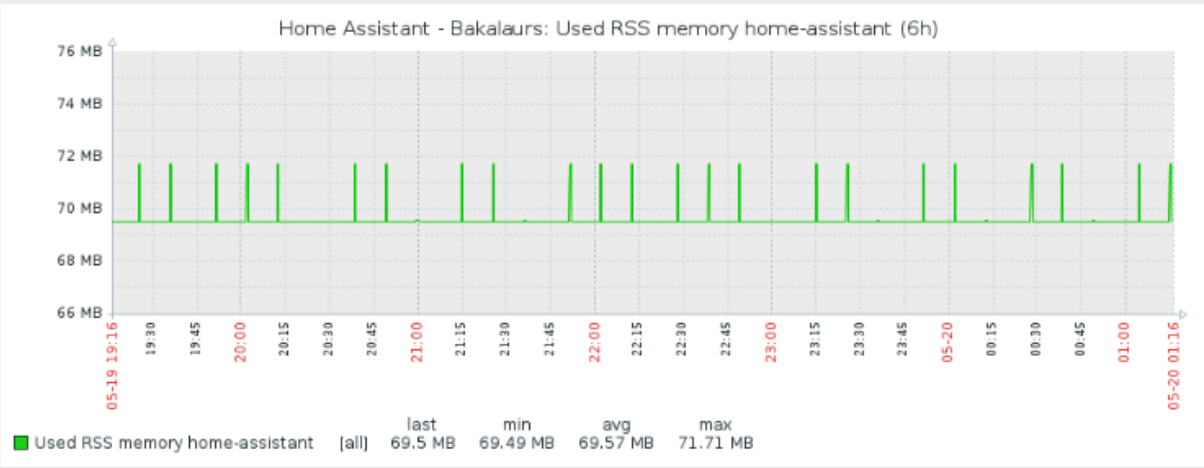
2.3.4. Home Assistant izmantotie sistēmas resursi

Veicot Home Assistant platformas CPU resursu novērošanu 6 stundu garumā, tika novērota attēlā 2.10 redzamā CPU noslodze.



Att. 2.10. **Home Assistant CPU noslodzes grafiks 6 stundu periodā**

Vidējā CPU noslodze bija 0.13% ar periodiskām CPU noslodzēm katru stundu līdz 0.85%.



Att. 2.11. **Home Assistant RAM RSS noslodzes grafiks 6 stundu periodā**

Home Assistant vidēja RAM RSS izmantojamība ir redzama attēlā 2.11 un liecina par ļoti mazu atmiņas izmantojamību, vidēji tiek izmantots 70MB RAM RSS atmiņas, kas periodiski palielinās līdz 72MB.

2.3.5. Home Assistant drošība

Home Assistant šobrīd neatbalsta funkcionalitātes ierobežošanu un vairāklietotāju atbalstu vienā platformas serverī, tomēr funkcionalitātes ieviešana tiek plānota [27]. Home Assistant šobrīd atbalsta viena lietotāja autentifikāciju, izmantojot paroli, kuru norāda YAML iestatījumu datnēs [28].

Iebūvētais MQTT papildinājums piedāvā izmantot gan lietotāja un paroles autentifikāciju ar MQTT serveri, gan izmantot klienta sertifikātus [22, 29].

Home Assistant iebūvēto papildinājumu drošība nav bijusi dziļi pētīta, tomēr fakts, ka tīmekļa lietotnei ir iespējams vienkārši uzstādīt autentifikāciju, būtiski samazina risku ļaunprātīgai platformas izmantošanai.

2.4. Platformu apkopojums

Platformu apkopojuma nodaļā OpenHAB un Home Assistant platformas tiek savstarpēji salīdzinātas, tādejādi ļaujot nonākt pie saderīgākās piedāvātajam risinājumam.

2.4.1. Kopienas aktivitāte

Salīdzinot OpenHAB un Home Assistant kopienas aktivitāti, var secināt, ka Home Assistant platformai ir lielāka kopienas aktivitāte un platformas lietotāji ir ieinteresēti uzlabot dokumentāciju. OpenHAB kopienas aktivitāte ir augsta, tomēr 4 reizes mazāka nekā Home Assistant.

2.4.2. Platformas dokumentācija

OpenHAB platformas dokumentācija ir sarežģītāka, lielākoties paša OpenHAB izvēlētās arhitektūras dēļ, kurā ir jānorāda ne tikai vienības, bet arī to vietnes kartes (sitemaps), papildus tam, dokumentācijā ir nepilnīgi atspoguļotas problēmsituācijas, kamēr Home Assistant ir veltītas atsevišķas sadaļas problēmu risināšanai. Home Assistant izvēlētā arhitektūra būtiski atvieglo platformas uzstādīšanu, un tiek izmantots plaši pazīstams YAML formāts, kas būtiski atvieglo iestatījumu izveidi.

2.4.3. Daudzlietotāju atbalsts

Abas platformas nepiedāvā iebūvētu funkcionalitāti, lai atbalstītu vairākus, savstarpēji atdalītus lietotājus, tomēr Home Asssistant būtiski atvieglo veidņu izveidi, jo atbalsta vides mainīgos. OpenHAB nepiedāvā iespēju automātiski atklāt jaunus resursus, izmantojot MQTT, kamēr Home Assistant ir iebūvēta MQTT atklāšanas funkcionalitāte.

2.4.4. Izmantotie sistēmas resursi

Home Assistant izmanto līdz pat 4 reizēm mazāk sistēmas RSS RAM resursu nekā OpenHAB, galvenokārt, jo OpenHAB ir izstrādāts, izmantojot Java programmēšanas valodu, un izmanto JVM, savukārt Home Asssistant ir izstrādāts, izmantojot Python programmēšanas valodu.

2.4.5. Drošība

OpenHAB drošības problēmas ir tikušas pētītas un aprakstītas, galvenokārt problēmas sagādā fakts, ka OpenHAB papildinājumiem netiek veikta integritātes pārbaude un netiek piedāvāta iebūvēta funkcionalitāte, lai veiktu lietotāja autentifikāciju. Home Assistant piedāvā iespēju veikt lietotāja autentifikāciju tīmekļa vietnē.

3. PIEDĀVĀTAIS RISINĀJUMS

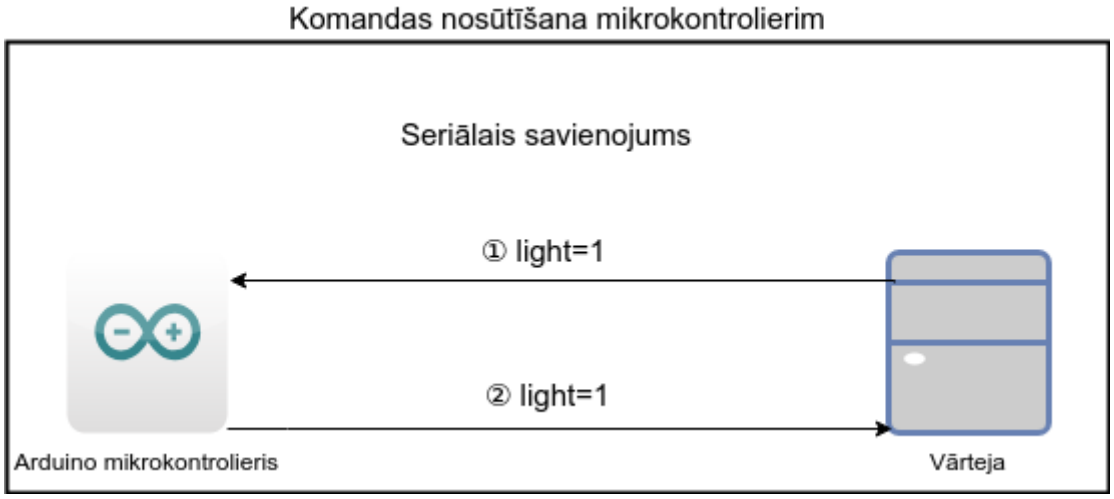
Ņemot vērā iepriekš apskatīto platformu apskatu, tika izvēlēta Home Assistant platforma.

Arhitektūra kopumā sastāv no 4-5 slāņiem, kurā katrai ir noteikts mērķis. Lai veiktu platformas realizāciju, tika izmantots 5 slāņu princips, kurā papildus mikrokontrolierim, internetam, mākoņplatformai un platformai ir vārteja. Gadījumā, kad mikrokontrolierim ir tieša pieeja internetam un ir pieejams MQTT protokola atbalsts, vārteja var nebūt nepieciešama.

Piedāvātā risinājuma grafisks pārskats ir apskatāms 4. pielikumā.

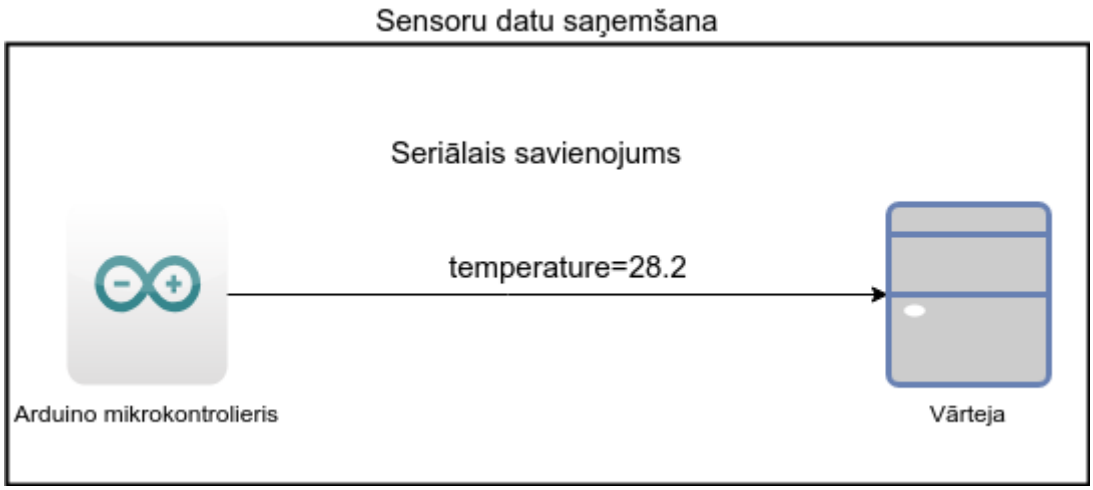
3.1. Mikrokontrolieris

Par mikrokontrolieri tiek izmantots Arduino nano, kurš ievāc temperatūras, mitruma un augstuma informāciju no BMP280 un SI7021 sensoriem un pārraida sensoru statusu, izmantojot seriālo savienojumu uz vārteju. Lai atvieglotu ierīces pārsūtīto datu parsēšanu, tiek izmantots vienība:vērtība (key-value) formāts. Mikrokontroliera izmantotais pirmkods ir pieejams 5. pielikumā.



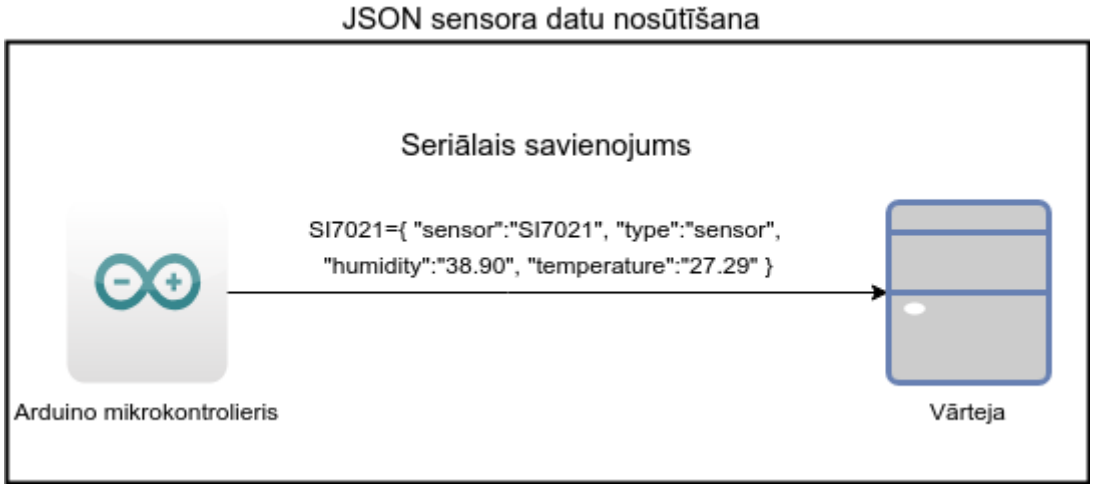
Att. 3.1. **Komandas nosūtīšana mikrokontrolieriem**

Mikrokontrolierim saņemot komandu no vārtejas, tiek izpildīta dotā komanda, nododot to sensoram. Ja komandas izpilde ir noritējusi veiksmīgi, tad mikrokontrolieris atbild ar jauno sensora statusu. Attēlā 3.1 ir parādīts vienkārša gaismas slēdža darbības princips un savstarpējā komunikācija ar mikrokontrolieri un vārteju.



Att. 3.2. **Vērtības saņemšana no mikrokontroliera**

Mikrokontrolieris reizi 3 sekundēs nolasa temperatūras, mitruma, augstuma informāciju no SI7021 un BMP280 sensoriem un pārraida to, izmantojot seriālo savienojumu uz vārteju. Attēlā 3.2 ir atspoguļots temperatūras nolasīšanas process un tā pārraide vārtejai.



Att. 3.3. **JSON vērtības nosūtīšana no mikrokontroliera**

Sensoru dati tiek sūtīti arī vienība:vērtība formātā, kur vērtība ir JSON formātā, kas atvieglo Home Assistant MQTT atklāšanas implementāciju. Lai nodrošinātu JSON formāta atbalstu Arduino, tiek izmantota ArduinoJson pakotne. Attēlā 3.3 ir parādīta JSON vērtības nosūtīšana mikrokontrolierim.

Lai varētu nodrošināt MQTT atklāšanas mehānisma darbību, mikrokontrolieris sūta iestatījumu informāciju par tam pieejamajiem sensoriem un Home Assistant iestatījumiem reizi 60 sekundēs, JSON formātā. Lai atšķirtu iestatījumu informāciju no sensoru vērtībām, tā tiek sūtīta ar "/config" affiksu. Funkcionalitātes implementāciju var apskatīt 5. pielikuma "write_config" funkcijā.

3.2. Vārteja

Vārtejas implementācija ir izstrādāta, izmantojot Python programmēšanas valodu, un tas var tikt palaists uz jebkura Linux bāzētā OS. Vārtejas pirmkods ir apskatāms 6. pielikumā. Vārteja nodrošina datu saņemšanu no mikrokontroliera, izmantojot seriālo interfeisu, pēdējo saņemto MQTT komandu nosūtīšanu mikrokontroliem, saņemto datu apstrādi un nosūtīšanu uz MQTT serveri. Izsaucot vārtejas programmu, tās argumentos tiek norādīts seriālais savienojums, no kura lasīt datus, klienta identifikators, MQTT servera atrašanās vieta un, ja nepieciešams, MQTT lietotājvārds un parole.

Saņemot sensoru informāciju no mikrokontroliera, vārteja veic saņemtā sensora informācija prefiksēšanu ar klienta identifikatoru, kurš bija norādīts, palaižot programmu un "/state" afiksēšanu, tādejādi uzlabojot MQTT tēmu lasāmību - izdalot komandas un statusus.

Saņemot sensora iestatījumu informāciju no mikrokontroliera, vārteja veic sekojošās darbības:

- Noņem "device_class" JSON vērtību un saglabā to mainīgajā, kas tiek izmantots, lai konstruētu Home Assistant MQTT atklāšanas tēmu. Galvenokārt tēmas noņemšana tiek veikta, jo sākot ar Home Assistant 0.69.1 versiju, "device_class" ir kļuvis par izmantojamu mainīgo Home Asssistant sensoram, bet "device_class" sūtīta informācija neatspoguļo vērtību, kas var tikt izmantot sensora iestatījumiem. Tā var tikt izmantota tikai, lai konstruēta MQTT tēmu [30];
- Papildina "state_topic" vērību ar lietotāja identifikatoru un pievieno "/state" virkni. Vērtība "state_topic" tiek izmantota, lai norādītu, kurā MQTT tēmā Home Assistant saņems sensora vērtību;
- Izmantojot "device_class" mainīgo, tiek noteikts, vai saņemtā informācija pieder gaismas sensoram. Gadījumā, kad informācija pieder gaismas sensoram, tiek pievienota "command_topic" vērtība JSON objektam, kas Home Assistant norāda tēmu, kurā ir jāsūta komandas, lai kontrolētu gaismas sensoru;
- Pievieno klienta identifikatoru un virkni "homeassistant/" tēmas sākumā, tādejādi norādot, ka tā ir Home Assistant MQTT atklāšanas tēma;
- Nosūta konstruēto tēmu un tās vērtību uz MQTT serveri.

3.3. Home Assistant platforma

Nodaļā ir aprakstītas Home Assistant platformas, resursdatora sastāvdaļas un to uzstādīšanas specifika.

3.3.1. Daudzlietotāju uzturēšana

Kopumā, lai nodrošinātu daudzlietotāju uzturēšanu, tiek izmantoti sekojošie servisi:

- Docker – ļauj atdalīt lietotāju Home Assistant instances izmantojot konteinerus;
- MySQL – dalīts resurss, kurā tiek uzglabāti Home Assistant instanču vēsturiskie dati;
- Mosquitto serveris – dalīts resurss, kurš tiek izmantots MQTT ziņapmaiņai ar vārteju un Home Assistant instanci.

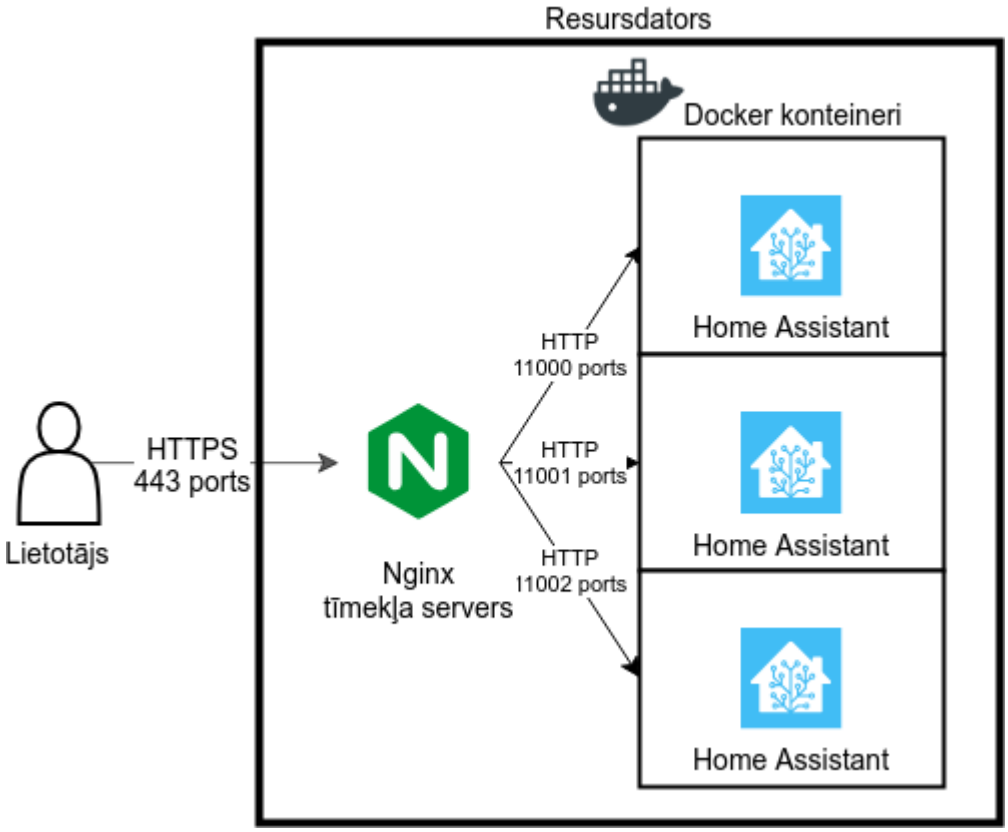
Servisu sākotnējas uzstādīšanas skripts ir apskatāms 7. pielikumā.

Home Assistant platformai nav iebūvēts atbalsts vairāk kā vienam atdalītam lietotājam, kā rezultātā katram lietotājam tiek atdalīta Home Assistant instance, izmantojot Docker rīku. Viens no galvenajiem Docker rīka lietošanas iemesliem ir iespēja nodrošināt drošību, gadījumā, kad lietotāja izmantotā Home Assistant lietotne instancei ir drošības politikas pārkāpums (compromised). Tiek būtiski samazināta iespēja uzbrucējam izkļūt no konteinera un tādejādi inficēt citas instances, papildus tam Docker piedāvā iespēju ierobežot konteinera pieeju resursdatora RAM un CPU resursiem [31, 32].

Katram lietotājam tiek izveidota mājas direktorija, kurā tiek uzglabāti Home Assistant iestatījumi. Palaižot lietotāja Docker konteineri, konteinerim tiek noteikta Home Assistant mājas iestatījumu direktorija, vides mainīgie, kuri tiek izmantoti Home Assistant iestatījumiem un atvēlēts brīvs ports uz resursdatora. Lai nodrošinātu sākotnējo veidņu izveidi, tiek klonēta Git repozitorija ar Home Assistant iestatījumiem, kura satur atsauces uz vides mainīgajiem.

Lai uzlabotu veiktspēju un dotu iespēju Home Asssistant lietotājiem izmantot HTTPS, uz resursdatora ir uzstādīts Nginx tīmekļa serveris, kurš terminē SSL savienojumu un nodod HTTP savienojumu Docker konteinerim. Tā kā katram lietotājam ir atvēlēts savs domēns, nav iespējams izmantot viena domēna SSL sertifikātu un ir nepieciešams automatizēt SSL sertifikātu iegūšanu. Lai iegūtu sertifikātus, tika izvēlēts Letsencrypt serviss, kurš piedāvā uzticamus SSL sertifikātus. Izstrādājot platformu, katram lietotājam tika izmantots viens SSL sertifikāts, tomēr, testējot platformu, tika nonākts pie secinājuma, ka šāda pieeja ierobežotu maksimālo jauno lietotāju skaitu līdz 20 nedēļā [33]. Problēma tika risināta, pārveidojot SSL

sertifikātu iegūšanu. Tā vietā, lai izmantotu katram lietotājam savu SSL sertifikātu, tiek izmantots aizstājējzīmes (wildcard) SSL sertifikāts, kurš aptver visu domēnu. Lai izmantotu aizstājējzīmes sertifikātu, ir nepieciešams veikt domēna piederības pārbaudi, kas tiek veikta, izmantojot certbot rīka Route 53 papildinājuma palīdzību. Tā kā Letsencrypt SSL sertifikāti ir derīgi tikai 3 mēnešus, resursdatorā tiek norādīts, ka reizi dienā ir jāmēģina atjaunot sertifikāts.



Att. 3.4. **Tīmekļa servera savienojuma nodošana Docker konteineriem**

Izveidojot katru lietotāju, tiek automātiski izveidoti lietotāja Nginx iestatījumi, norādot Docker konteinera portu un faktu, ka ir jāveic SSL terminēšana. Attēlā 3.4 ir atspoguļota lietotāja savienojuma nodošana Docker konteineriem.

3.3.2. Veidņu izmantošana

Platformas uzstādīšana var būt sarežģīts process, kurā ir jāveic Home Assistant iestatījumu datņu rediģēšana. Lai šo procesu atvieglotu, Home Assistant iestatījumu datnē tiek norādīts, ka MQTT klienta, MQTT atklāšanai, MySQL datubāzes savienojumam, laika zonai ir jāatsaucas uz vides mainīgajos esošajām vērtībām. Vides mainīgie tiek norādīti, izveidojot lietotāju, un no tiem izveidota vides mainīgo datne, kas tiek nodota Docker konteinerim. Palaižot Home Assistant, tiks automātiski aizpildītas vides mainīgo vērtības. Vides mainīgo izveides un nodošanas process ir atspoguļots 9. pielikumā

Būtiska sistēmas sastāvdaļa ir iespēja veikt Home Assistant MQTT atklāšanu. Lai Home Assistant automātiski veiktu atklāšanu, tam YAML iestatījumu datnē ir jāieslēdz un jānorāda MQTT atklāšanas prefikss. Atklāšanas prefikss tiek automātiski izveidots lietotāja izveides procesa laikā un, līdzīgi kā citi iestatījumi, nodots Home Assistant, izmantojot vides mainīgos. Attēlā 3.5 ir parādīti Home Assistant iestatījumi, lai izmantotu MQTT atklāšanu ar vides mainīgajiem.

```
mqtt:
    discovery: true
    discovery_prefix: !env_var HA_MQTT_DISCOVER_PREFIX
```

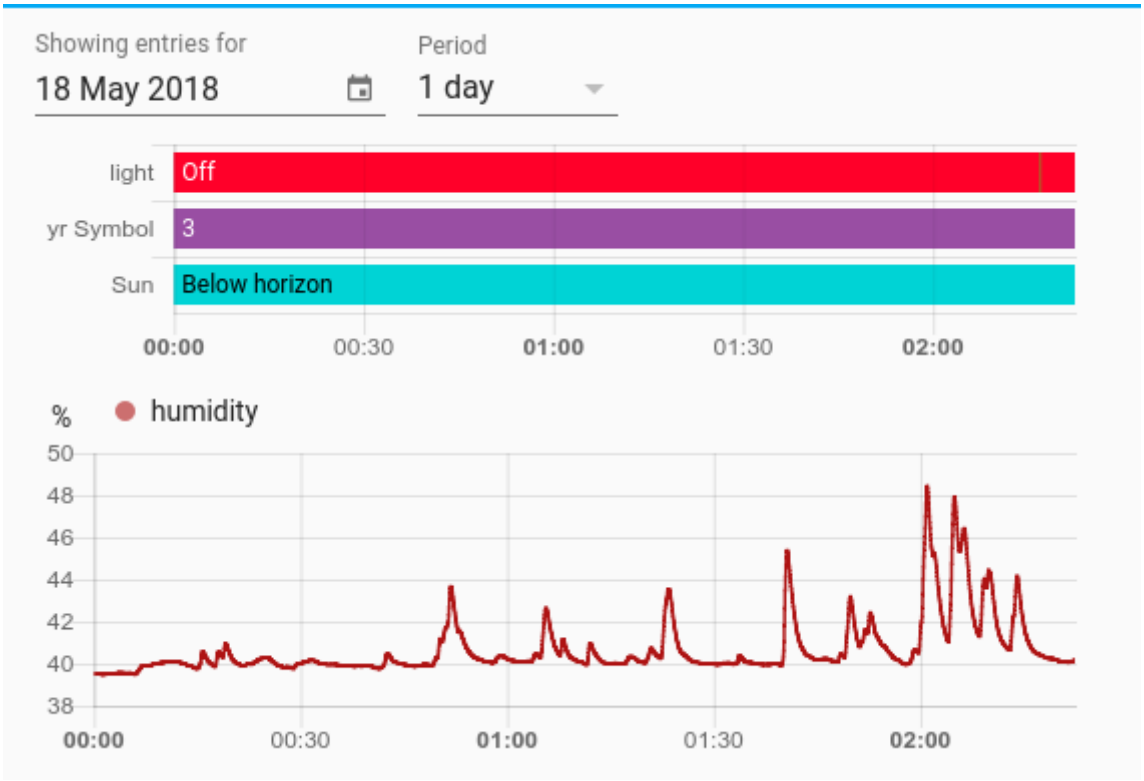
Att. 3.5. **Home Assistant MQTT atklāšanas iestatījumi izmantojot vides mainīgos**

Pēc platformas testēšanas tika novērots, ka Home Assistant neatbalsta prefiksus, kas satur "/" simbolu, kā rezultātā ir jāizmanto lietotāja identifikators un neatdalīta simbolu virkne, lai norādītu MQTT atklāšanas tēmu [25]. Lietotāju uzstādīšanas skripts ir apskatāms 8. pielikumā.

3.3.3. Datu vizualizācija

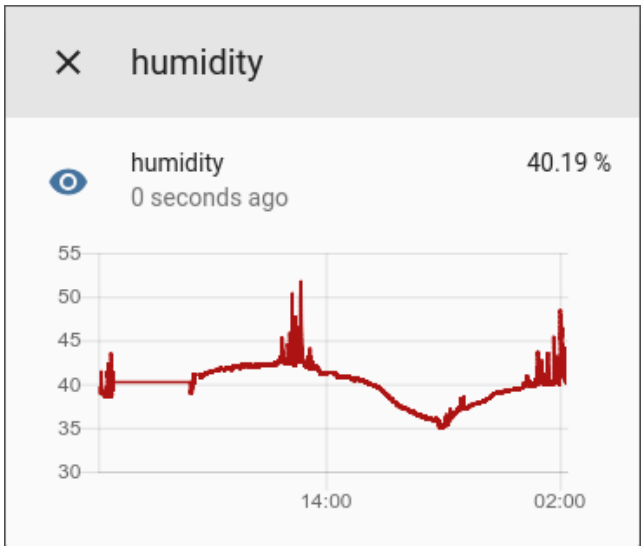
Lai nodrošinātu vēsturisko datu vizualizāciju, uz resursdatora ir uzstādīta MySQL datubāze, kurā Home Assistant uzglabā vēsturiskās sensoru mērījumu vērtības. Home Assistant iestatījumos, izmantojot vides mainīgos, ir norādīts, ka jāizmanto MySQL datubāze.

Home Assistant piedāvā iespēju veikt datu vizualizāciju tīmekļa vietnē bez iepriekšējas iestatījumu datņu maiņas. Tā ir apskatāma zem "History" sadaļas vai arī informācijas panelī (dashboard).



Att. 3.6. **Home Assistant "History" paneļa vienības vizualizācija**

"History" panelis var atspoguļot laika periodus, sākot ar dienu, un iekļauj visu vienību mērījumus. Paneļa vizualizāciju var redzēt attēlā 3.6.



Att. 3.7. **Home Assistant informācijas paneļa vienības vizualizācija**

Informācijas panelī ir iespējams apskatīt mazu vienības vēsturisko apkopojumu pēdējai dienai. Informācijas paneļa vienības vizualizācija ir redzama attēlā 3.7.

Papildus iepriekš minētajai funkcionalitātei Home Assistant piedāvā iespēju veikt komplicētāku grafiku uzstādīšanu, tomēr iestatījumu definēšana ir jāveic iestatījumu datnēs, jo tika novērots, ka MQTT atklāšana neatbalsta iespēju izveidot iepriekš definētus, pielāgotas vēstures vizualizācijas [34].

3.3.4. Drošība

Lai nodrošinātu izstrādātas platformas drošību un neļautu platformas klientiem piekļūt pie savstarpējiem dalītajiem resursiem, tiek izmantota dalītajos resursos pieejamā funkcionalitāte, kas ierobežo pieeju resursiem.

Katram no platformas lietotājiem ir izdalīta sava MySQL datubāze ar lietotāju un paroli. Home Assistant MySQL lietotāji nevar piekļūt pie citu lietotāju datubāzēm. Izveidojot jaunu platformas lietotāju, tiek automātiski izveidota MySQL datubāze ar lietotāju un paroli. Lai nodrošinātu šādu funkcionalitāti, tiek izmantota MySQL iebūvētie lietotāju un pieejas kontroles mehānismi [35].

Lai nodrošinātu dalītu MQTT serveri un samazināto izmantojamo resursu daudzumu, katram lietotājam tiek izmantots dalīts Mosquitto serveris. Mosquitto piedāvā iespēju izmantot lietotājvārdus un paroles, kā arī autorizēt piekļuvi noteiktām MQTT tēmām, izmantojot ACL. Mosquitto ir iebūvēta funkcionalitāte, kura ļauj pārvaldīt piekļuvi, ieskaitot lietotājvārdus un paroles, izmantojot datnes [36]. Pēc piedāvāto datņu sintakses un funkcionalitātes apskates tika apskatīti citi risinājumi, kas ļautu izmantot citu veidu Mosquitto autentifikācijas un autorizācijas datu uzglabāšanu, galvenokārt tāpēc, ka iebūvēto MQTT datņu mehānismu ir sarežģīti kontrolēt mainīgā daudzlietotāju vidē.

Implementējot MQTT autentifikāciju, tika izvēlēts Mosquitto mosquitto-auth-plug papildinājums, kurš ļauj izmantot MySQL datubāzi, lai uzglabātu lietotāju datus un pieejas politikas [37]. Lai izmantotu papildinājumu, dalītajā datubāzes serverī tika uzstādīta jauna datubāze un datubāzes lietotājs, kurš ļauj Mosquitto rīkam veikt autentifikācijas un autorizācijas datu nolasi no datubāzes. Papildus iepriekš minētajam, lietotāja izveides skripts tika papildināts ar funkcionalitāti, kura pievieno jaunu lietotāju un šifrētu paroli MySQL datubāzē, kā arī izveido sekojošās Mosquitto pieejas politikas:

- ļaut lasīt-rakstīt piekļuvi MQTT tēmām, kuras sākās ar lietotājvārdu;
- ļaut lasīt-rakstīt piekļuvi MQTT tēmām, kuras sākās ar lietotājvārdu apvienotu ar virkni "homeassistant".

```
INSERT INTO acls (username, topic, rw) VALUES
('${USRNAME}', '${USRNAME}/#', 2);

  INSERT INTO acls (username, topic, rw) VALUES
('${USRNAME}', '${USRNAME}homeassistant/#', 2);
```

Att. 3.8. **Datubāzē izveidotās Mosquitto pieejas politikas**

Attēlā 3.8 ir parādīti SQL vaicājumi, ar kuriem ACL tiek ievietotas datubāzē. # simbols ir aizstājējzīme (wildcard), kas apraksta, visas turpmākās MQTT apakštēmas [36].

Lai apgrūtinātu iespēju lietotāju Home Assistant instancēm, kurām ir drošības politikas pārkāpums (compromised), izmanto Docker konteineri. Docker konteineri būtiski apgrūtina iespēju veikt drošības politikas pārkāpumu uz resursdatora un ietekmēt citu platformas lietotāju darbību [32].

Iespēja izveidot šifrētu savienojumu ar tīmekļa vietni ir būtiska mūsdienu sastāvdaļa, kā rezultātā tiek uzstādīts Nginx tīmekļa serveris ar Letsencrypt SSL sertifikātu, tādejādi nodrošinot šifrētu savienojumu ar platformu.

3.3.5. Rezerves kopijas

Uz platformas servera eksistē sekojošie dati, kuru saglabāšana ir būtiska platformas lietošanai:

- Home Assistant iestatījumu datnes;
- MySQL datubāzes ar Mosquitto autentifikācijas un autorizācija informāciju, kā arī platformas lietotāju veiktajiem mērījumiem.

Lai nodrošinātu Home Assistant iestatījumu datņu rezerves kopijas un MySQL datubāzes rezerves kopijas, tiek izmantots skripts, kurš veic sekojošās darbības:

- arhivē lietotāju mājas direktorijas ar tajās iekļautajām Home Assistant iestatījumu datnēm un Docker izmantotajiem vides mainīgajiem;
- veic datubāzes datu saglabāšanu datnē, izmantojot mysqldump rīku [38];
- augšupielādē izveidotās rezerves kopijas uz AWS S3 servisu.

Rezerves kopiju izveides skripts ir apskatāms 10. pielikumā. Skripts ir pievienots resursdatoram, un norādīts, ka tam ir jāizpildās reizi dienā, nakts stundās.

AWS S3 pakalpojums ir uzstādīts, izmantojot dzīvescikla politiku (lifecycle policy), kas norāda, ka visas rezerves kopijas pēc 30 dienām tiek pārvietotas uz retas pieejas klasi (infrequent access), lai samazinātu rezerves kopiju uzglabāšanas izmaksas, un pilnībā jādzēš pēc 3 mēnešiem [39].

REZULTĀTI

Darbā ir izpētītas lietu interneta arhitektūras un apskatīta OpenHAB un Home Assistant lietu interneta platformu funkcionalitāte. Apskatīto platformu funkcionalitāte ir savstarpēji salīdzināta, tādejādi ļaujot izvēlēties piemērotāko platformu, kurai ir iespējams implementēt daudzlietotāju atbalstu un kuru piedāvātās funkcionalitātes dēļ ir iespējams piedāvāt kā pakalpojumu.

Pateicoties apskatīto platformu salīdzinājumam, ir izveidots platformas arhitektūras risinājums, kurš ļauj veikt vienkāršu sākotnējo platformas uzstādīšanu, nodrošina daudzlietotāju atbalstu, izmantojot vairākas platformas instances vienā resursdatorā, un parūpējas par lietotāju datu drošību. Lai automatizētu platformas servera uzstādīšanu, ir izveidoti lietotāju izveides un sākotnējās servera uzstādīšanas skripti.

Platformas klientu iekārtu uzstādīšanas atvieglošanai ir izveidots Arduino un vārtejas pirmkoda paraugs, kurš parūpējas par Home Assistant iestatījumu automātisku izveidi, izmantojot platformā iebūvēto MQTT sensoru atklāšanas funkcionalitāti.

SECINĀJUMI

Veicot OpenHAB platformas apskati, tika secināts, ka platformas kopiena ir vidēji aktīva, platforma nepiedāvā iespēju vienkārši automatizēt jaunu vienību izveidi un platformas iestatījumus nav iespējams izveidot, izmantojot platformā iebūvētus mehānismus, kā rezultātā ir jāizmanto operētājsistēmas rīki, lai veiktu iestatījumu datņu izveidi. OpenHAB drošība ir tikusi pētīta, un tās papildinājumi ne vienmēr piedāvā drošāko risinājumu, papildus tam, OpenHAB platformai nav iebūvētas funkcionalitātes, lai autentificētu lietotājus.

Home Assistant platformas apskates laikā tika secināts, ka platforma pievērš pastiprinātu uzmanību dokumentācijai un platformas kodola arhitektūra būtiski atvieglo sākotnējo platformas uzstādīšanu, vadoties pēc dokumentācijas, papildus tam dokumentācijā arī ir ietverti MQTT funkcionalitātes testēšanas soļi. Home Assistant ir iebūvēta funkcionalitāte, kura ļauj izveidot veidnes no iestatījumu datnēm, pateicoties iespējai atsaukties uz sistēmas vides mainīgajiem, veidojot iestatījumu datnes. Home Assistant piedāvā vienkāršu un efektīvu lietotāja autentifikāciju.

Veicot platformas arhitektūras izveidi, tika secināts, ka Home Assistant iebūvētā MQTT atklāšanas funkcionalitāte nodrošina vienkāršu veidu, kā atklāt jaunus sensorus, tomēr tās funkcionalitāte ir limitēta – tā ļauj pievienot tikai sensoru informāciju un jaunu, komplicētu sensoru vienību grafiku pievienošana nav iespējama.

IZMANTOTĀ LITERATŪRA UN AVOTI

[1] I. Yaqoob, E. Ahmed, I. Hashem, A. Gani, M. Imran un M. Guizani, «A vision of IoT: Applications, challenges, and opportunities with china perspective,» *2012 10th International Conference on Frontiers of Information Technology (FIT): Proceedings*, sēj. 1, nr. 4, pp. 349-359, 2012.

[2] J. Gubbi, R. Buyya, S. Marusic un M. Palaniswami, «Internet of Things (IoT): A vision, architectural elements, and future directions,» *Future generation computer systems*, sēj. 29, nr. 8, pp. 1645-1660, 2013.

[3] J. Chase, «The evolution of the internet of things,» 2013.. [Tiešsaiste]. Pieejams: http://www.ti.com/lit/ml/swrb028/swrb028.pdf. [Piekļūts 28. 04. 2018.].

[4] J. Rodriguez, «Building an IOT Platform: Centralized vs. Decentralized Models | Jesus Rodriguez | Pulse | LinkedIn,» 16. 06. 2015.. [Tiešsaiste]. Pieejams: https://www.linkedin.com/pulse/building-iot-platform-centralized-vs-decentralized-models-rodriguez. [Piekļūts 28. 04. 2018.].

[5] I. Yaqoob, E. Ahmed, I. Hashem, I. A. T. Hashem, A. Gani, M. Imran un M. Guizani, «Internet of things architecture: Recent advances, taxonomy, requirements, and open challenges,» *IEEE wireless communications*, sēj. 24, nr. 3., pp. 10-16, 2017.

[6] Cisco, «Simplified IoT Architecture. Cisco Netacad,» [Tiešsaiste]. Pieejams: https://static-course-assets.s3.amazonaws.com/IoECT2/en/index.html#1.2.1.5. [Piekļūts 8. 05. 2018.].

[7] K. Kreuzer, «openHAB @ Eclipse DemoCamp Darmstadt,» 14. 07. 2010. [Tiešsaiste]. Pieejams: https://www.slideshare.net/xthirtynine/openhab-eclipse-democamp-darmstadt. [Piekļūts 28. 04. 2018.].

[8] «Bindings - openHAB 2 - Empowering the Smart Home,» [Tiešsaiste]. Pieejams: https://docs.openhab.org/addons/bindings.html. [Piekļūts 14. 05. 2018.].

[9] «openHAB - Introduction,» [Tiešsaiste]. Pieejams: https://www.openhab.org/introduction.html. [Piekļūts 15. 05. 2018.].

[10] «openHAB 2 - Empowering the Smart Home,» [Tiešsaiste]. Pieejams: https://docs.openhab.org/introduction.html. [Piekļūts 15. 05. 2018.].

[11] «Concepts - openHAB 2 - Empowering the Smart Home,» [Tiešsaiste]. Pieejams: https://docs.openhab.org/concepts/index.html. [Piekļūts 5. 05. 2018.].

[12] B. Gilmer, «User-friendly solution for plug-in debugging · Issue #299 · openhab/openhab-core · GitHub,» 19. februāris 2018.. [Tiešsaiste]. Pieejams: https://github.com/openhab/openhab-core/issues/299. [Piekļūts 2. 05. 2018.].

[13] «MQTT - Bindings - openHAB 2 - Empowering the Smart Home,» [Tiešsaiste]. Pieejams: https://docs.openhab.org/addons/bindings/mqtt1/readme.html. [Piekļūts 2. 05. 2018.].

[14] «Securing Communication and Access - openHAB 2 - Empowering the Smart Home,» [Tiešsaiste]. Pieejams: https://docs.openhab.org/installation/security.html. [Piekļūts 5. 05. 2018].

[15] «openHAB2: connect to MQTT using client certificate,» [Tiešsaiste]. Pieejams: https://community.openhab.org/t/openhab2-connect-to-mqtt-using-client-certificate/25251. [Piekļūts 12. 05. 2018.].

[16] M. Ramljak, «Security analysis of Open Home Automation Bus system,» %1 *Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2017 40th International Convention on*, Opatija, 2017.

[17] «Trivia - Home Assistant,» [Tiešsaiste]. Pieejams: https://www.home-assistant.io/help/trivia. [Piekļūts 7. 05. 2018.].

[18] «Components - Home Assistant,» [Tiešsaiste]. Pieejams: https://www.home-assistant.io/components/. [Piekļūts 12. 05. 2018.].

[19] «YAML - Home Assistant,» [Tiešsaiste]. Pieejams: https://www.home-assistant.io/docs/configuration/yaml/. [Piekļūts 12. 05. 2018.].

[20] «home-assistant/home-assistant | Coveralls - Test Coverage History; Statistics,» [Tiešsaiste]. Pieejams: https://coveralls.io/github/home-assistant/home-assistant. [Piekļūts 12. 05. 2018].

[21] «Architecture · Home Assistant dev docs,» [Tiešsaiste]. Pieejams: https://developers.home-assistant.io/docs/en/architecture_index.html. [Piekļūts 12. 05. 2018.].

[22] «MQTT Brokers - Home Assistant,» [Tiešsaiste]. Pieejams: https://www.home-assistant.io/docs/mqtt/broker. [Piekļūts 12. 05. 2018.].

[23] «MQTT Testing - Home Assistant,» [Tiešsaiste]. Pieejams: https://www.home-assistant.io/docs/mqtt/testing/. [Piekļūts 12. 05. 2018.].

[24] «Recorder - Home Assistant,» [Tiešsaiste]. Pieejams: https://www.home-assistant.io/components/recorder/. [Piekļūts 12. 05. 2018.].

[25] «MQTT Discovery - Home Assistant,» [Tiešsaiste]. Pieejams: https://www.home-assistant.io/docs/mqtt/discovery/. [Piekļūts 12. 05. 2018.].

[26] E. Nilsson, «GitHub - voxic/ha-editor: A web-based editor for homeassistant based on the Monaco editor,» [Tiešsaiste]. Pieejams: https://github.com/voxic/ha-editor. [Piekļūts 12. 05. 2018.].

[27] «Users & Authentication · Issue #17 · home-assistant/architecture · GitHub,» [Tiešsaiste]. Pieejams: https://github.com/home-assistant/architecture/issues/17. [Piekļūts 12. 05. 2018.].

[28] «Setup basic information - Home Assistant,» [Tiešsaiste]. Pieejams: https://www.home-assistant.io/docs/configuration/basic/. [Piekļūts 13. 05. 2018.].

[29] «MQTT Certificate - Home Assistant,» [Tiešsaiste]. Pieejams: https://www.home-assistant.io/docs/mqtt/certificate/. [Piekļūts 13. 05. 2018.].

[30] «(Another?) MQTT issue on 0.69.0b2 · Issue #14360 · home-assistant/home-assistant · GitHub,» [Tiešsaiste]. Pieejams: https://github.com/home-assistant/home-assistant/issues/14360. [Piekļūts 15. 05. 2018.].

[31] «Limit a container's resources | Docker Documentation,» [Tiešsaiste]. Pieejams: https://docs.docker.com/config/containers/resource_constraints/. [Piekļūts 18. 05. 2018.].

[32] T. Bui, «Analysis of docker security,» *CoRR,* sēj. abs/1501.02967, 2015..

[33] «Rate Limits - Let's Encrypt - Free SSL/TLS Certificates,» [Tiešsaiste]. Pieejams: https://letsencrypt.org/docs/rate-limits/. [Piekļūts 18. 05. 2018.].

[34] «History Graph - Home Assistant,» [Tiešsaiste]. Pieejams: https://www.home-assistant.io/components/history_graph/. [Piekļūts 18. 05. 2018.].

[35] «MySQL 5.6 Reference Manual :: 6.3.2 Adding User Accounts,» [Tiešsaiste]. Pieejams: https://dev.mysql.com/doc/refman/5.6/en/adding-users.html. [Piekļūts 18. 05. 2018.].

[36] R. Light, «mosquitto.conf man page | Eclipse Mosquitto,» [Tiešsaiste]. Pieejams: https://mosquitto.org/man/mosquitto-conf-5.html. [Piekļūts 18. 05. 2018.].

[37] «GitHub - jpmens/mosquitto-auth-plug: Authentication plugin for Mosquitto with multiple back-ends (MySQL, Redis, CDB, SQLite3),» [Tiešsaiste]. Pieejams: https://github.com/jpmens/mosquitto-auth-plug. [Piekļūts 18. 05. 2018.].

[38] «MySQL 5.6 Reference Manual :: 4.5.4 mysqldump — A Database Backup Program,» [Tiešsaiste]. Pieejams: https://dev.mysql.com/doc/refman/5.6/en/mysqldump.html. [Piekļūts 18. 05. 2018.].

[39] Amazon Web Services, «How Do I Create a Lifecycle Policy for an S3 Bucket? - Amazon Simple Storage Service,» [Tiešsaiste]. Pieejams: https://docs.aws.amazon.com/AmazonS3/latest/user-guide/create-lifecycle.html. [Piekļūts 19. 05. 2018.].

PIELIKUMI

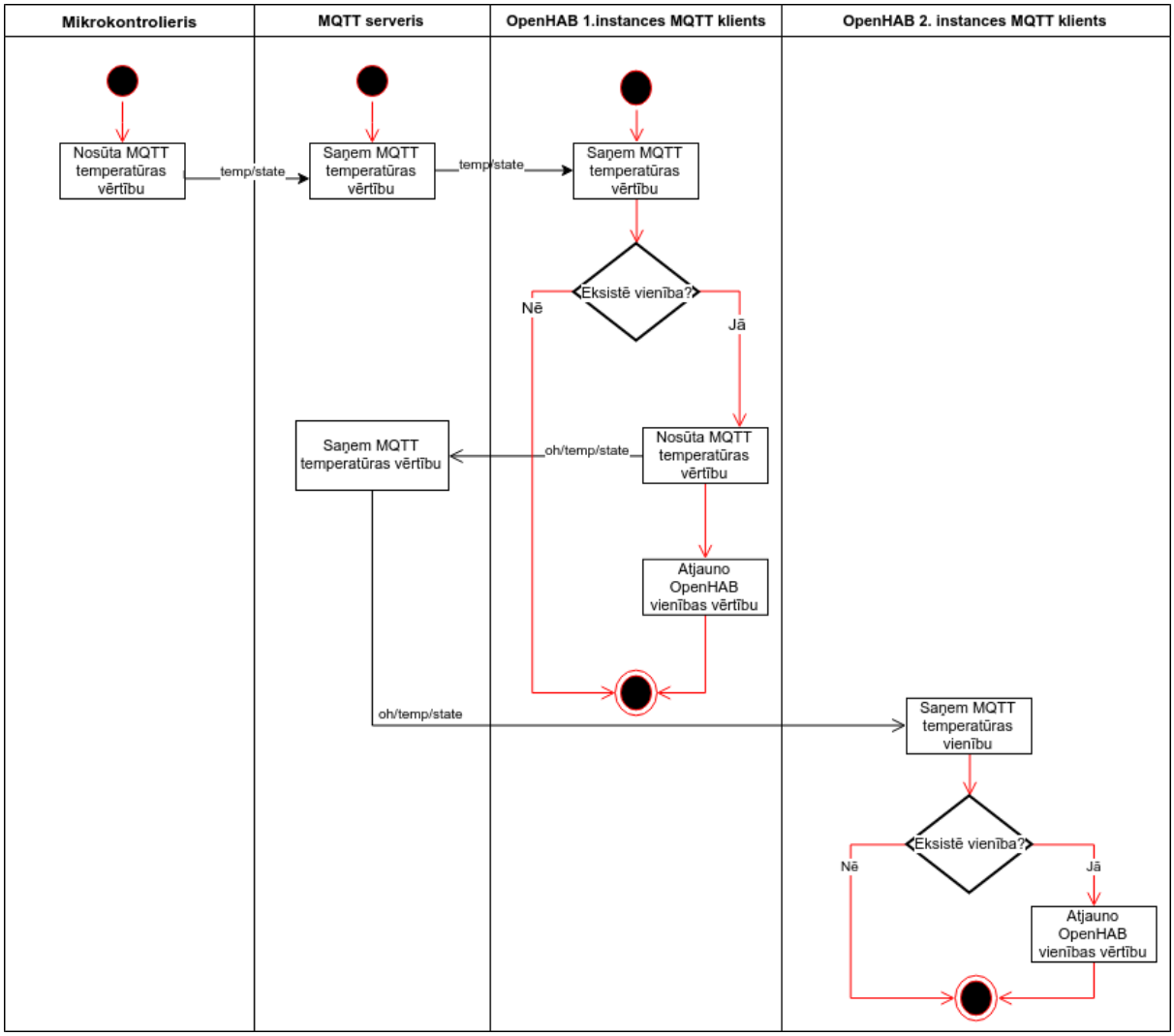
1. pielikums

OpenHAB MQTT servisa iestatījumi

```
#
# Define your MQTT broker connections here for use in the MQTT Binding or
MQTT
# Persistence bundles. Replace <broker> with an ID you choose.
#

# URL to the MQTT broker, e.g. tcp://localhost:1883 or ssl://localhost:8883
mqtt.url=tcp://172.17.0.1:1883

# Optional. Client id (max 23 chars) to use when connecting to the broker.
# If not provided a random default is generated.
mqtt.clientId=test

# Optional. True or false. If set to true, allows the use of clientId
values
# up to 65535 characters long. Defaults to false.
# NOTE: clientId values longer than 23 characters may not be supported by
all
# MQTT servers. Check the server documentation.
#<broker>.allowLongerClientIds=false

# Optional. User id to authenticate with the broker.
#mqtt.user=<user>

# Optional. Password to authenticate with the broker.
#<broker>.pwd=<password>

# Optional. Set the quality of service level for sending messages to this
broker.
# Possible values are 0 (Deliver at most once),1 (Deliver at least once) or
2
# (Deliver exactly once). Defaults to 0.
#<broker>.qos=<qos>

# Optional. True or false. Defines if the broker should retain the messages
sent to
# it. Defaults to false.
mqtt.retain=false
```

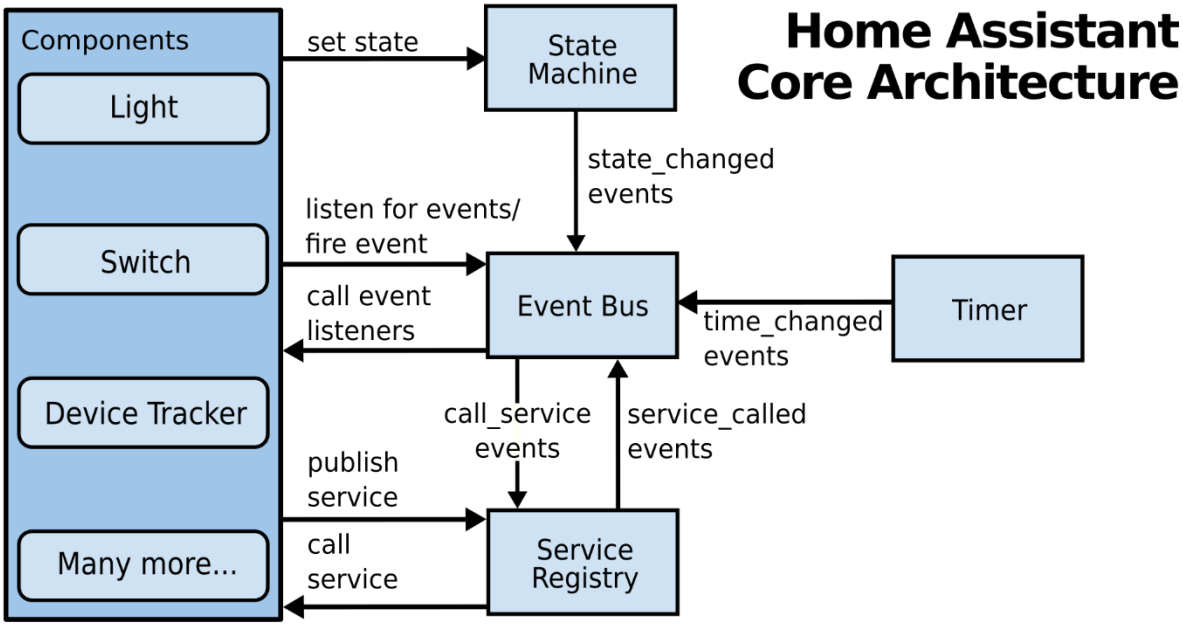
2. pielikums

OpenHAB MQTT EventBus paredzētā izmantošana



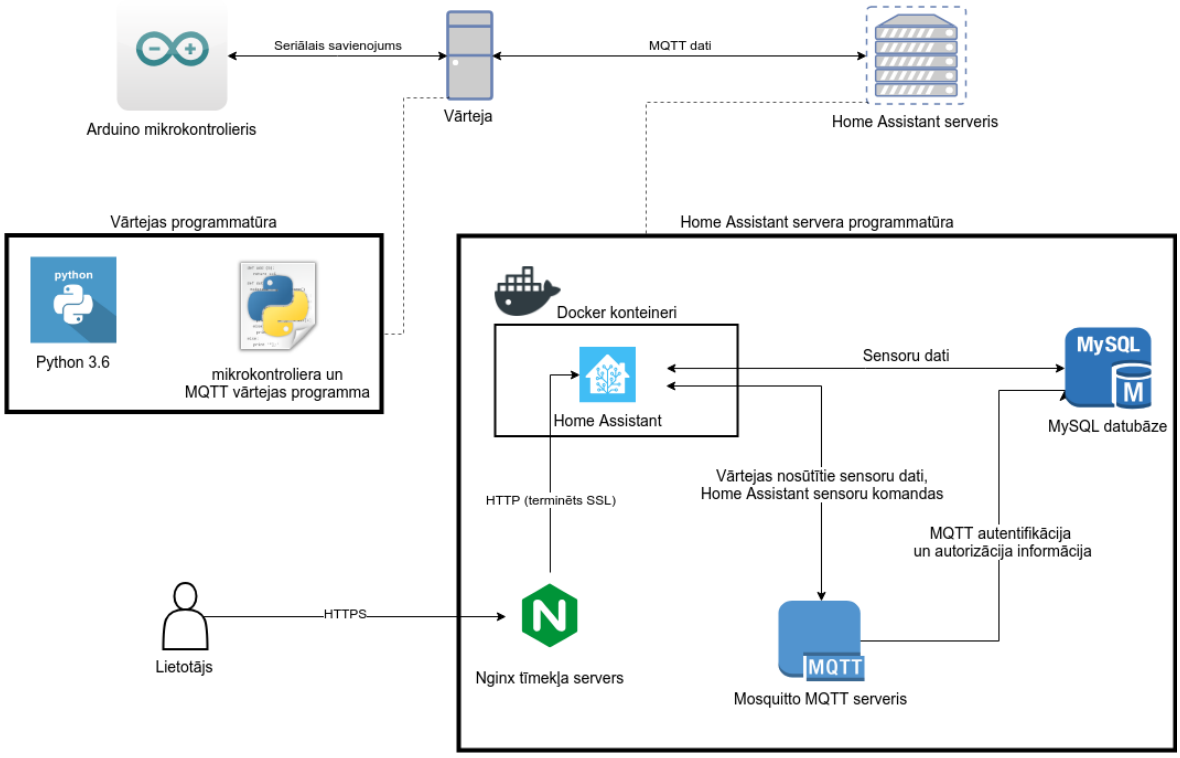
3. pielikums

Home Assistant kodola savstarpējo procesu reprezentācija [20]



4. pielikums

Izstrādātās platformas arhitektūra



Mikrokontroliera pirmkods

```
#include < SPI.h >
#include < Wire.h >

#include "i2c.h"

#include "i2c SI7021.h"
#include "i2c_BMP280.h"

#include < ArduinoJson.h >

SI7021 si7021;
BMP280 bmp280;
byte irsi = 0, irbmp = 0;
unsigned long previousSensorMillis = 0;
unsigned long previousUptimeMillis = 0;
unsigned long previousConfigMillis = 0;
unsigned long previousLightMillis = 0;
unsigned long previousLightStateMillis = 0;
const unsigned configInterval = 60000;
const unsigned sensorInterval = 3000;
const unsigned uptimeInterval = 1000;
const unsigned lightInterval = 500;
const unsigned lightStateInterval = 10000;
bool onboard_light_state = false;

void setup() {
  Serial.begin(9600);

  Serial.print("Probe SI7021: ");
  if (si7021.initialize()) {
    Serial.println("SI7021 found!");
    irsi = 1;
  } else {
    Serial.println("SI7021 missing");
    irsi = 0;
  }

  Serial.print("Probe BMP280: ");
  if (bmp280.initialize()) {
    Serial.println("BMP280 found");
    irbmp = 1;
  } else {
    Serial.println("BMP280 missing");
    irbmp = 0;
  }

  // onetime-measure:
  bmp280.setEnabled(0);
  bmp280.triggerMeasurement();

  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, LOW);
}

void write_value(String key, char *
  const value) {
  Serial.print(key);
  Serial.print("=");
  Serial.println(value);
}

void write config(JsonObject & state) {
  StaticJsonBuffer < 160 > jsonBuffer;

  for (auto kv: state) {

    if (kv.key != "sensor" && kv.key != "type") {
      JsonObject & config = jsonBuffer.createObject();
      config["name"] = kv.key;
      config["device_class"] = state.get < String > ("type");
      if (kv.key == "light") {
        config["payload_on"] = "1";
```

```
        config["payload_off"] = "0";
        config["state_topic"] = state.get < String > ("sensor") + "_" + state.get < String >
("type"); // Prefixed out by consumer
      } else {
        char tmp[40];
        sprintf(tmp, "{{ value_json.%s }}", kv.key);
        config["value_template"] = tmp;
        config["state_topic"] = state.get < String > ("sensor"); // Prefixed out by consumer
        if (kv.key == "temperature") config["unit_of_measurement"] = "C";
        else if (kv.key == "humidity") config["unit_of_measurement"] = "%";
        else if (kv.key == "altitude") config["unit_of_measurement"] = "m";
        else if (kv.key == "pressure") config["unit_of_measurement"] = "psi";
      }
      char jsonChar[160];
      config.printTo(jsonChar);

      char key[40];
      sprintf(key, "%s%s/config", state.get <
        const char * > ("sensor"), kv.key);
      write_value(key, jsonChar);
      jsonBuffer.clear();
    }

  }
}

struct KeyValue {
  String key;
  int value;
  bool initialised;
};

struct KeyValue get_line() {
  struct KeyValue data;
  data.initialised = false;
  if (Serial.available() > 0) {
    data.key = Serial.readStringUntil('=');
    data.value = Serial.readStringUntil('\n').toInt();
    data.initialised = true;
  }
  return data;
}

void loop() {
  static float humi, temp;
  float temperature;
  float pascal;
  static float meters, metersold;
  unsigned long currentMillis = millis();
  StaticJsonBuffer < 150 > jsonBuffer;
  char jsonChar[150];
  char tmp[20];

  if (currentMillis - previousUptimeMillis >= uptimeInterval) {
    previousUptimeMillis = currentMillis;
    sprintf(tmp, "%d", millis() / 1000);
    write_value("uptime", tmp);
  }

  if (currentMillis - previousSensorMillis >= sensorInterval) {
    previousSensorMillis = currentMillis;

    si7021.getHumidity(humi);
    si7021.getTemperature(temp);
    si7021.triggerMeasurement();

    dtostrf(temp, 0, 2, tmp);
    write_value("SI7021_temperature", tmp);
    dtostrf(humi, 0, 2, tmp);
    write_value("SI7021_humidity", tmp);

    JsonObject & si7021_state = jsonBuffer.createObject();
    si7021_state["sensor"] = "SI7021";
    si7021_state["type"] = "sensor";
    si7021_state["humidity"] = String(humi);
    si7021_state["temperature"] = String(temp);

    si7021_state.printTo(jsonChar);
    write_value("SI7021", jsonChar);
```

```
    // Write configuration every x seconds and on first sensor interval
    if (currentMillis - previousConfigMillis >= configInterval || currentMillis <=
sensorInterval + 1000) {
      write_config(si7021_state);
    }

    jsonBuffer.clear();

    bmp280.awaitMeasurement();
    bmp280.getTemperature(temperature);
    bmp280.getPressure(pascal);
    bmp280.getAltitude(meters);

    metersold = (metersold * 10 + meters) / 11;
    bmp280.triggerMeasurement();

    dtostrf(temperature, 0, 2, tmp);
    write_value("BMP280_temperature", tmp);
    dtostrf(meters, 0, 2, tmp);
    write_value("BMP280_altitude", tmp);
    dtostrf(metersold, 0, 2, tmp);
    write_value("BMP280_altitude_pt1", tmp);
    dtostrf(pascal, 0, 2, tmp);
    write_value("BMP280_pressure", tmp);

    //Write also JSON values
    JsonObject & bmp280_state = jsonBuffer.createObject();
    bmp280_state["sensor"] = "BMP280";
    bmp280_state["type"] = "sensor";
    bmp280_state["altitude"] = String(meters);
    bmp280_state["pressure"] = String(pascal);
    bmp280_state["temperature"] = String(temperature);

    bmp280_state.printTo(jsonChar);
    write_value("BMP280", jsonChar);

    // Write configuration every x seconds and on first sensor interval
    if (currentMillis - previousConfigMillis >= configInterval || currentMillis <=
sensorInterval + 1000) {
      write_config(bmp280_state);
    }

    if (currentMillis - previousConfigMillis >= configInterval) {
      previousConfigMillis = currentMillis;
    }

    jsonBuffer.clear();

  }

  // Every Light State interval write its state
  if (currentMillis - previousLightStateMillis >= lightStateInterval || currentMillis <= 100)
{
    previousLightStateMillis = currentMillis;
    JsonObject & light_state = jsonBuffer.createObject();
    light_state["sensor"] = "onboard";
    light_state["type"] = "light";
    light_state["light"] = String(onboard_light_state);
    light_state.printTo(jsonChar);
    char tmp2[20];
    sprintf(tmp, "%s", light_state.get <
      const char * > ("light"));
    sprintf(tmp2, "%s_%s", light_state.get <
      const char * > ("sensor"), light_state.get <
        const char * > ("type"));
    write_value(tmp2, tmp);
    write_value(light_state.get < String > ("sensor"), jsonChar);

    write_config(light_state);

    jsonBuffer.clear();

  }

  if (currentMillis - previousLightMillis >= lightInterval) {
    previousLightMillis = currentMillis;
    // Light default state on boot is OFF
    // Take care of light boot
```

```
    struct KeyValue data = get_line();

    if (data.initialised == true && data.key == "onboard_light") {
      JsonObject & light_state = jsonBuffer.createObject();
      light_state["sensor"] = "onboard";
      light_state["type"] = "light";

      if (data.value == 1) {
        write_value(data.key, "1");
        light_state["light"] = "1";
        digitalWrite(LED_BUILTIN, HIGH);
        onboard_light_state = true;
      } else {
        write_value(data.key, "0");
        light_state["light"] = "0";
        digitalWrite(LED_BUILTIN, LOW);
        onboard_light_state = false;
      }

      light_state.printTo(jsonChar);
      write_value(light_state.get < String > ("sensor"), jsonChar);

    }
  }

  delay(100);
}
```

Vārtejas Python pirmkods

```
import serial
import paho.mqtt.client as mqtt
import time
import re
import json
import logging
import argparse
import sys

# The callback for when the client receives a CONNACK response from the server.
def on_connect(client, userdata, flags, rc):
    print("Connected with result code "+str(rc))
    if rc != 0:
        sys.exit("MQTT connection error")
    # Subscribing in on_connect() means that if we lose the connection and
    # reconnect then subscriptions will be renewed.
    client.subscribe(userdata.client_id + "/+/command")
    file_init_state()

# The callback for when a PUBLISH message is received from the server.
def on_message(client, userdata, msg):
    print("Received: " + msg.topic+" " +str(msg.payload.decode()))
    key = msg.topic.split("/")[1]
    value = msg.payload.decode()
    ser.write(str.encode(key +"="+value +"\r\n"))
    file_save_state(key, value)

def on_disconnect(client, userdata, rc):
    print("Connection lost")

def parse_line(line):
    pattern = re.compile("(?P<key>.+)=(?P<value>.+)\r")
    data = pattern.match(line)
    if data:
        return data.groupdict()
    else:
        return None

def file_get_state():
    try:
        with open('state.json', 'r') as f:
            state = json.load(f)

    except(FileNotFoundError):
        print("Creating statefile")
        with open('state.json', 'w+') as f:
            state = {}
            json.dump(state, f)
    return state

def file_save_state(key, value):
    state = file_get_state()
    state[key] = value
    with open('state.json', 'w') as f:
        json.dump(state, f)

def file_init_state():
    print("Initializing saved states")
    state = file_get_state()
    print(state)
    for key, value in state.items():
        print("sent to serial: " + key +"="+value)
        ser.write(str.encode(key +"="+value +"\r\n"))
```

```
def mqtt_send_state(mqtt_client, type, state, client_id):
    mqtt_client.publish(client_id + "/"+type+"/state", payload=state, qos=1,
retain=True)

def mqtt_send_config(mqtt_client, type, config, client_id):
    configJson = json.loads(config)
    # device_class is required only for publish topic, we do not need to pass it
forward
    device_class = configJson.pop('device_class', None)
    # Lights can receive commands
    if device_class == "light":
        configJson["command_topic"] = client_id + "/" + configJson["state_topic"] +
"/command"
    configJson["state_topic"] = client_id + "/" + configJson["state_topic"]
+"/state"
    # Discovery prefix cannot include / and as such temporary measure is required
until the bug is resolved
    # https://github.com/home-assistant/home-assistant/issues/14360
    mqtt_client.publish(client_id + "homeassistant/" + device_class + "/" + type,
payload=json.dumps(configJson), qos=1, retain=True)

def open_serial(serial_interface):
    ser = serial.Serial(serial_interface, 9600, timeout=15)
    return ser

def read_serial(ser, mqtt_client, client_id):
    file_init_state()
    mqtt_client.loop_start()
    while True:
        line = ser.readline().decode("windows-1252")
        data = parse_line(line)
        if data is not None:
            if 'config' in data["key"]:
                mqtt_send_config(mqtt_client, data["key"], data["value"],
client_id)
            else:
                mqtt_send_state(mqtt_client, data["key"], data["value"], client_id)


def parse_arguments():
    parser = argparse.ArgumentParser(
        description='MQTT Gateway',
        formatter_class=argparse.RawDescriptionHelpFormatter)
    parser.add_argument(
        '--host', dest='host',
        action='store',
        required=True
    )
    parser.add_argument(
        '-u', '--username', dest='username',
        action='store',
        required=False,
        help='MQTT username'
    )
    parser.add_argument(
        '-p', '--password', dest='password',
        action='store',
        required=False,
        help='MQTT password'
    )
    parser.add_argument(
        '--port', dest='port',
        action='store',
        default=1883,
        type=int,
        required=False,
        help='MQTT port'
    )
    parser.add_argument(
        '-s', '--serial', dest='serial',
```

```
        action='store',
        required=False,
        default="/dev/ttyUSB0",
        help = 'Serial device location'
    )
    parser.add_argument(
        '--client-id', dest='client_id',
        action='store',
        required=False,
        default="test",
        help = 'Client ID used in MQTT topics'
    )
    return parser.parse_args()


def init_mqtt_client(args):
    client = mqtt.Client()
    client.enable_logger()
    client.on_connect = on_connect
    client.on_message = on_message
    client.on_disconnect = on_disconnect

    if args.username or args.password:
        client.username_pw_set(args.username, args.password)

    client.connect(args.host, args.port, 60)
    client.user_data_set(args)
    return client

def init_logging(client):
    logging.basicConfig(level=logging.INFO)
    logger = logging.getLogger()
    client.enable_logger(logger)


if __name__ == "__main__":
    args = parse_arguments()
    mqtt_client = init_mqtt_client(args)
    init_logging(mqtt_client)
    ser = open_serial(args.serial)
    read_serial(ser, mqtt_client, args.client_id)
```

Platformas sākotnējās uzstādīšanas skripts

```
#!/bin/bash
set -euxo pipefail

if [[ $EUID -ne 0 ]]; then
   echo "This script must be run as root"
   exit 1
fi

source .server-env

export DEBIAN_FRONTEND=noninteractive

# Helper functions
is_mysql_installed(){
  dpkg-query -W -f='${Status}' mysql-server-5.6 2>/dev/null | grep -c "ok
installed"
}

# Server setup

update_server(){
  apt-get update
  apt-get upgrade -y
  apt-get install vim ncdu htop vnstat iftop tmux -y
}

setup_mysql() {
  apt-get install software-properties-common -y
  add-apt-repository -y ppa:ondrej/mysql-5.6 -y
  apt-get update
  apt-get -y install mysql-server-5.6
  sed -i 's/.*bind-address.*/bind-address = 0.0.0.0/g'
/etc/mysql/mysql.conf.d/mysqld.cnf
  service mysql restart
}

configure_mysql(){
  mv /root/.my.cnf /root/.my.cnf.old ||:
  mysqladmin -u root password ${DB_PASSWORD}
  cat >/root/.my.cnf <<EOF
[client]
user=root
password=${DB_PASSWORD}
EOF
}

setup_mosquitto(){
  apt-get install mosquitto mosquitto-clients -y
}

setup_docker(){
  curl -fsSL https://download.docker.com/linux/ubuntu/gpg | apt-key add -
  add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu
$(lsb_release -cs) stable"
  apt-get update
  apt-get install -y docker-ce
}

setup_letsencrypt(){
    add-apt-repository ppa:certbot/certbot -y
    apt-get update
    apt-get install software-properties-common -y
    apt-get install python-certbot-nginx -y
    apt-get install python-pip -y
```

```
    pip install certbot_dns_route53
}

letsencrypt_run(){
    stat /etc/letsencrypt/live/$DOMAIN/fullchain.pem || certbot certonly --
noninteractive --agree-tos -m $EMAIL --dns-route53 --server https://acme-
v02.api.letsencrypt.org/directory -d $DOMAIN -d *.$DOMAIN
}

letsencrypt_persist(){
    echo "09 1 * * * /usr/bin/letsencrypt renew --renew-hook \"/bin/systemctl
restart nginx\"" > /etc/cron.d/le-$DOMAIN

}

setup_nginx(){
  sudo apt-get install nginx apache2-utils -y
}

configure_nginx(){
    stat /etc/ssl/certs/dhparam.pem || openssl dhparam -out
/etc/ssl/certs/dhparam.pem 2048


    mkdir /etc/nginx/global || true
    cat >/etc/nginx/global/ssl-validation <<EOF
            location ~ /.well-known/acme-challenge/ {
            allow all;
                    root /var/www/ssl-validation/;
            }
EOF
    mkdir /var/www/ssl-validation/ || true


    cat >/etc/nginx/sites-enabled/default <<EOF
server {
        server_name _;
        listen 80;
        listen [::]:80;
        location / {
                return          301 https://\$http_host\$request_uri;
        }
        include global/ssl-validation;
}

server {
    server_name *.$DOMAIN $DOMAIN;
       listen 443 ssl;
       listen [::]:443 ssl;

    ssl_certificate /etc/letsencrypt/live/$DOMAIN/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/$DOMAIN/privkey.pem;
ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
    ssl_prefer_server_ciphers on;
    ssl_dhparam /etc/ssl/certs/dhparam.pem;
    ssl_ciphers 'ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-
RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-
DSS-AES128-GCM-SHA256:kEDH+AESGCM:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-
SHA256:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-
ECDSA-AES256-SHA384:ECDHE-RSA-AES256-SHA:ECDHE-ECDSA-AES256-SHA:DHE-RSA-AES128-
SHA256:DHE-RSA-AES128-SHA:DHE-DSS-AES128-SHA256:DHE-RSA-AES256-SHA256:DHE-DSS-
AES256-SHA:DHE-RSA-AES256-SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:AES128-
SHA256:AES256-SHA256:AES128-SHA:AES256-SHA:AES:CAMELLIA:DES-CBC3-
SHA:!aNULL:!eNULL:!EXPORT:!DES:!RC4:!MD5:!PSK:!aECDH:!EDH-DSS-DES-CBC3-SHA:!EDH-
RSA-DES-CBC3-SHA:!KRB5-DES-CBC3-SHA';
    ssl_session_timeout 1d;
    ssl_session_cache shared:SSL:50m;
    ssl_stapling on;
    ssl_stapling_verify on;
    location / {
```

```
        return 404;
    }
    include global/ssl-validation;
}
EOF
    service nginx restart
}

setup_dockbix(){
    docker rm dockbix-agent-xxl -f || true
    docker run \
      --name=dockbix-agent-xxl \
      --net=host \
      --privileged \
      -v /:/rootfs \
      -v /var/run:/var/run \
      --restart unless-stopped \
      -e "ZA_Server=${ZABBIX_IP}" \
      -e "ZA_ServerActive=${ZABBIX_IP}" \
      -d monitoringartist/dockbix-agent-xxl-limited:latest
}

setup_mosquitto_db(){
  apt-get install whois -y
  mysql <<EOF
  GRANT USAGE ON *.* TO 'mosquitto'@'127.0.0.1';
  DROP USER 'mosquitto'@'127.0.0.1';
  create database if not exists mosquitto;
  CREATE USER 'mosquitto'@'127.0.0.1' IDENTIFIED BY '${MOSQUITTO_DB_PASSWORD}';
  GRANT SELECT ON mosquitto.* TO 'mosquitto'@'127.0.0.1';
EOF

  rm -rf /tmp/auth || true
  git clone https://github.com/jpmens/mosquitto-auth-plug.git -b 0.1.2 /tmp/auth
  cd /tmp/auth
  cp config.mk.in config.mk
  make np
  cp np /usr/local/bin/np || true


    mysql <<EOF
use mosquitto;

CREATE TABLE IF NOT EXISTS users (
        id INTEGER AUTO_INCREMENT,
        username VARCHAR(25) NOT NULL,
        pw VARCHAR(255) NOT NULL,
        super INT(1) NOT NULL DEFAULT 0,
        PRIMARY KEY (id)
  );

set @x := (select count(*) from information_schema.statistics where table_name =
'users' and index_name = 'users_username' and table_schema = database());
set @sqlx := if( @x > 0, 'select ''Index exists.''', 'Alter Table users ADD UNIQUE
INDEX users_username (username);');
PREPARE stmtx FROM @sqlx;
EXECUTE stmtx;

DELETE FROM users WHERE username LIKE '${MOSQUITTO_SUPERUSER_NAME}';
INSERT INTO users (username, pw, super) VALUES ('${MOSQUITTO_SUPERUSER_NAME}', '$(
np -p $MOSQUITTO_SUPERUSER_PASSWORD)', '1');


CREATE TABLE IF NOT EXISTS acls (
        id INTEGER AUTO_INCREMENT,
        username VARCHAR(25) NOT NULL,
        topic VARCHAR(256) NOT NULL,
        rw INTEGER(1) NOT NULL DEFAULT 1,       -- 1: read-only, 2: read-write
        PRIMARY KEY (id)
        );
```

```
set @y := (select count(*) from information_schema.statistics where table_name =
'acls' and index_name = 'acls_user_topic' and table_schema = database());
set @sqly := if( @y > 0, 'select ''Index exists.''', 'Alter Table acls ADD UNIQUE
INDEX acls_user_topic (username, topic(228));');
PREPARE stmty FROM @sqly;
EXECUTE stmty;

EOF
}

setup_mosquitto_auth(){
  apt-get install mosquitto mosquitto-clients build-essential mosquitto-auth-plugin
libssl-dev -y

    cat >/etc/mosquitto/conf.d/auth.conf<<EOF
connection_messages true
allow_anonymous false
auth_plugin /usr/lib/mosquitto-auth-plugin/auth-plugin.so
auth_opt_backends mysql
auth_opt_host 127.0.0.1
auth_opt_port 3306
auth_opt_dbname mosquitto
auth_opt_user mosquitto
auth_opt_pass $MOSQUITTO_DB_PASSWORD
auth_opt_userquery SELECT pw FROM users WHERE username = '%s'
auth_opt_superquery SELECT COUNT(*) FROM users WHERE username = '%s' AND super = 1
auth_opt_aclquery SELECT topic FROM acls WHERE (username = '%s') AND (rw >= %d)
auth_opt_superusers $MOSQUITTO_SUPERUSER_NAME
auth_opt_cacheseconds 0
EOF
    cat >/etc/mosquitto/conf.d/log.conf<<EOF
log_type all
log_type debug
EOF
  service mosquitto restart
}



update_server
if ! is_mysql_installed; then
  setup_mysql
  configure_mysql
fi
setup_mosquitto
setup_docker
setup_nginx
configure_nginx
setup_letsencrypt
letsencrypt_run
letsencrypt_persist
setup_dockbix
setup_mosquitto_db
setup_mosquitto_auth
```

8. pielikums

Platformas lietotāju uzstādīšanas skripts

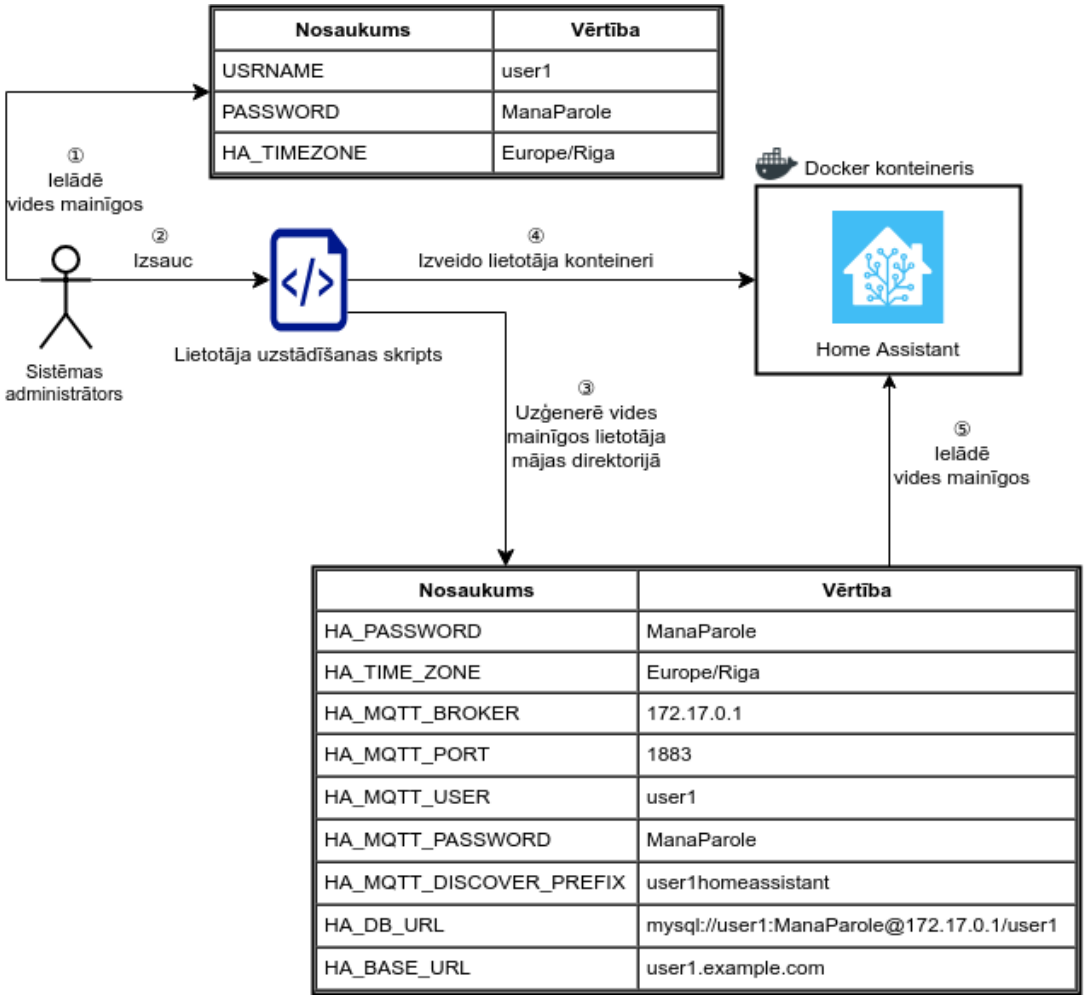
```
#!/bin/bash
set -ex

display_usage() {
      echo "This script must be run with super-user privileges."
      echo -e "\nUsage:\n$0 username [password] \n"
      }

if [ $EUID -ne 0 ]
then
      display_usage
      exit 1
fi

# check whether user had supplied -h or --help . If yes display usage
if [[ ( $# == "--help") ||  $# == "-h" ]]
then
      display_usage
      exit 0
fi



USRNAME=${USRNAME:-$1}
PASSWORD=${PASSWORD:-$(tr -cd '[:lower:]' < /dev/urandom | fold -w8 | head
-n1)}

source .user-env

create_user(){
  useradd -m -s /bin/nologin $USRNAME
  echo $USRNAME:$PASSWORD | /usr/sbin/chpasswd
  DOCKER_PORT=$(($(id -u $USRNAME) + $PORT_OFFSET))
  usermod -c $DOCKER_PORT $USRNAME
}

create_user_ha_env(){
cat > /home/$USRNAME/.ha-env <<EOF
    HA_PASSWORD=${HA_PASSWORD:-$PASSWORD}
    HA_TIME_ZONE=${HA_TIME_ZONE}
    HA_MQTT_BROKER=${HA_MQTT_BROKER}
    HA_MQTT_PORT=${HA_MQTT_PORT}
    HA_MQTT_USER=${HA_MQTT_USER:-$USRNAME}
    HA_MQTT_PASSWORD=${HA_MQTT_PASSWORD:-$PASSWORD}
    HA_MQTT_DISCOVER_PREFIX=${HA_MQTT_DISCOVER_PREFIX}
    HA_DB_URL=${HA_DB_URL}
    HA_BASE_URL=${HA_BASE_URL:-$BASE_DOMAIN}
    HA_MQTT_BIRTH_TOPIC=${HA_MQTT_BIRTH_TOPIC}
EOF

}

configure_user_nginx(){
```

```
cat >/etc/nginx/sites-enabled/site-$BASE_DOMAIN <<EOF
server {
    server_name $BASE_DOMAIN;
      listen 443 ssl;
      listen [::]:443 ssl;

    ssl_certificate /etc/letsencrypt/live/$DOMAIN/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/$DOMAIN/privkey.pem;
ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
    ssl_prefer_server_ciphers on;
    ssl_dhparam /etc/ssl/certs/dhparam.pem;
    ssl_ciphers 'ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-GCM-
SHA256:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES256-GCM-SHA384:DHE-RSA-
AES128-GCM-SHA256:DHE-DSS-AES128-GCM-SHA256:kEDH+AESGCM:ECDHE-RSA-AES128-
SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES128-
SHA:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES256-
SHA:ECDHE-ECDSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES128-SHA:DHE-
DSS-AES128-SHA256:DHE-RSA-AES256-SHA256:DHE-DSS-AES256-SHA:DHE-RSA-AES256-
SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:AES128-SHA256:AES256-SHA256:AES128-
SHA:AES256-SHA:AES:CAMELLIA:DES-CBC3-
SHA:!aNULL:!eNULL:!EXPORT:!DES:!RC4:!MD5:!PSK:!aECDH:!EDH-DSS-DES-CBC3-
SHA:!EDH-RSA-DES-CBC3-SHA:!KRB5-DES-CBC3-SHA';
    ssl_session_timeout 1d;
    ssl_session_cache shared:SSL:50m;
    ssl_stapling on;
    ssl_stapling_verify on;


  location / {
    proxy_pass  http://127.0.0.1:$DOCKER_PORT;
    add_header 'Access-Control-Allow-Origin' '*';

    # Pass a bunch of headers to the downstream server, so they'll know
what's going on.
    proxy_set_header Host \$host;
    proxy_set_header X-Real-IP \$remote_addr;
    proxy_set_header X-Forwarded-For \$proxy_add_x_forwarded_for;
    proxy_set_header Upgrade \$http_upgrade;
    proxy_set_header Connection "upgrade";

    # Most web apps can be configured to read this header and understand
that the current session is actually HTTPS.
    proxy_set_header X-Forwarded-Proto https;

    # We expect the downsteam servers to redirect to the right hostname, so
don't do any rewrites here.
    proxy_redirect     off;
  }
  include global/ssl-validation;
}

EOF
  service nginx reload
}

configure_user_database(){
  mysql <<EOF
create database $USRNAME;
CREATE USER '$USRNAME'@'172.17.%.%' IDENTIFIED BY '$PASSWORD';
GRANT ALL PRIVILEGES ON $USRNAME.* TO '$USRNAME'@'172.17.%.%';
EOF
}
```

```
configure_user_mosquitto(){
  mysql <<EOF
  use mosquitto;
  INSERT INTO users (username, pw, super) VALUES ('${USRNAME}', '$( np -p
${HA_MQTT_PASSWORD:-$PASSWORD})', '0');
  INSERT INTO acls (username, topic, rw) VALUES ('${USRNAME}',
'${USRNAME}/#', 2);
  INSERT INTO acls (username, topic, rw) VALUES ('${USRNAME}',
'${USRNAME}homeassistant/#', 2);
EOF
}

configure_user_ha(){
    git clone $GIT_REPO $HA_BASE_DIR
    chown -R $USRNAME  $HA_BASE_DIR
}

run_user_ha(){
  docker run \
    -p 127.0.0.1:$DOCKER_PORT:8123 \
    -v $HA_BASE_DIR/config:/config \
    -v /etc/localtime:/etc/localtime:ro \
    -d \
    --name $USRNAME \
    --user $(id -u $USRNAME) \
    --env-file /home/$USRNAME/.ha-env \
    homeassistant/home-assistant:0.69.1
}


write_info(){
      # Go into subshell so umask doesn't persist
      (umask 077; echo "$(date --iso-8601=seconds) username:$USRNAME
password:$PASSWORD docker_port=$DOCKER_PORT" >> /var/log/ha_user_log.txt)
}


create_user
create_user_ha_env
configure_user_database
configure_user_mosquitto
configure_user_ha
run_user_ha
configure_user_nginx
write_info
```

9. pielikums

Lietotāja vides mainīgo nodošana Docker konteinerim



10. pielikums

Izstrādātās platformas rezerves kopiju skripts

```
#!/bin/bash
set -exu
current_date=$(date -u +"%FT%H%MZ")
backup_tmp_dir="/tmp/backup"
s3_bucket="example-backups"
mkdir -p  backup_tmp_dir || true
tar -cvzf $backup_tmp_dir/home-$current_date.tar.gz /home
mysqldump --all-databases | gzip > $backup_tmp_dir/db-$current_date.sql.gz
aws s3 cp  $backup_tmp_dir s3://$s3_bucket/iot --recursive
rm -rf $backup_tmp_dir
```

DOKUMENTĀRĀ LAPA

Bakalaura darbs „Lietu internets kā pakalpojums” izstrādāts Latvijas Universitātes Datorikas fakultātē.

Ar savu parakstu apliecinu, ka pētījums veikts patstāvīgi, izmantoti tikai tajā norādītie informācijas avoti un iesniegtā darba elektroniskā kopija atbilst izdrukai.

Autors: Alberts Saulītis _____

Rekomendēju/nerekomendēju darbu aizstāvēšanai

Vadītājs: docents, Dr. dat. Leo Trukšāns _____

Recenzente: asoc.prof., Dr. dat. Lelde Lāce

Darbs iesniegts Datorikas fakultātē 28.05.2018.

Dekāna pilnvarotā persona:

vecākā metodiķe Ārija Sproģe_____

Darbs aizstāvēts bakalaura gala pārbaudījuma komisijas sēdē

___.06.2018. prot. Nr. _____.

Komisijas sekretārs(-e): _____