Opinnäytetyö (AMK)

Tieto- ja viestintätekniikka

2019

Niklas Helin

MENDER-ETÄPÄIVITYS YOCTO-KÄÄNNÖKSEEN

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tieto-ja viestintätekniikka

2019 | 36 sivua

Niklas Helin

MENDER OTA PÄIVITYS YOCTO KÄÄNNÖKSEEN

IP- ja Bluetooth-osoitteellisten laitteiden määrä kasvaa jatkuvasti, minkä seurauksena niistä tulee yhä suositumpia hakkeroinnin kohteita. Laitteiden päivittämisellä pystytään paikkaamaan esim. haavoittuvuuksia. Työn tarkoituksena on tutkia miten Mender-ohjelmistolla voidaan päivittää laite etänä. Laitteeseen tehtiin Yocto-projektilla Linux-käyttöjärjestelmä, johon sisällytetään Menderin asiakasohjelmisto, jolla saadaan yhteys palvelimelle.

Työtä lähdettiin kehittämään siten, että ensin määriteltiin vaatimukset, kehityksen vaiheet ja tarvittavat komponentit. Tuloksena saatiin toimiva järjestelmä ja järjestelmän toimivuus testattiin päivittämällä ohjelmisto siten, että käyttöjärjestelmään lisättiin uusi toiminnallisuus. Lisäksi järjestelmälle tehtiin testi, jossa testattiin järjestelmän kestävyyttä siten, että laitteesta otettiin virta pois kesken päivityksen.

ASIASANAT:

sulautettu Linux, etäpäivitys, IoT

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information and Communications Technology

2019 | 36 pages

Niklas Helin

MENDER OTA UPDATE TO YOCTO BUILD

A Number of devices with an IP- and a Bluetooth addresses increases continually which makes them more popular targets to hackers. By updating the software some level of e.g. of vulnerabilities can be patched. The meaning of this thesis is to explore how a embedded device with Linux operating system can be updated remotely with Mender software. The software of a device is created with the Yocto Project which creates a Linux distribution where the Mender client is included which allows to connect to the Mender server. The steps taken to achieve the objective of this thesis were defining requirements, designing the process and finding the components. The outcome was working Yocto build and working connection between the Mender server and the client. The outcome of the Yocto build was proven by adding a new feature to the operating system by using the remote update. The robustness of the system was tested by making power failure during an update process.

KEYWORDS:

embedded Linux, remote update, IoT

SISÄLTÖ

KÄYTETYT LYHENTEET TAI SANASTO

Sanasto	
Bitbake	Yleinen tehtävien suorittamisen moottori
Mender	Avoimen lähdekoodin ohjelmisto päivityksen toteuttamiseen
Open Embedded	Yocto projektissa käytetty koontijärjestelmä
Raspberry Pi	Raspberry Pi Foundationin kehittämä yhden piirilevyn tietokone.
Yocto Project	Avoimen lähdekoodin kokoelma työkaluja sulautetun Linux käyttöjärjestelmän tekemiseen.

1 JOHDANTO

Opinnäytetyön tarkoituksena ja tavoitteena oli Yocto-projektilla, sulautetun Linux-laitteen etäpäivittäminen, Mender-ohjelmistolla. Seuraavilla esimerkeillä osoitetaan aiheen merkitys, miksi sen käsitteleminen on ajankohtaista ja hyödyllistä.

Tammikuun ensimmäinen päivä v. 2020 Kalifornian osavaltiossa tulee uusi laki SB-327 voimaan, jossa määritellään uudet tietoturva vähimmäisvaatimukset kytkettyihin laitteisiin, joita usein myös kutsutaan IoT-laitteiksi ja älylaitteeksi. [1]

ETSI (European Telecommunications Standards Institute) julkaisi teknisen spesifikaation 103 645, jonka tavoitteena on antaa ohjeet miten kytkettyjen laitteiden tieturvaa ja yksityisyyden suojaa voidaan parantaa. Spesifikaatti pitää sisällään 13 vaatimusta, joista kolmas vaatimus on, että laitteiden ohjelmisto tulisi olla turvallisesti päivitettävissä. Päivitettävyys mahdollistaa haavoittuvuuksien ja muiden ohjelmistovirheiden korjaamisen. [1]

GDPR muutti Euroopassa toimivien ohjelmistoalan toimijoiden tapoja ja paransi yksityisen ihmisen oikeuksia. On mahdollista, että GDPR:n tapainen laki tulee myös kytkettyihin laitteisiin, jolloin voittajia on ne jotka ovat alusta asti miettineet laitteidensa ohjelmistoarkkitehtuurin. [1]

Edellä mainitut asiat ovat hyvä perusta sille, miksi opinnäytetyössä tutkittu laitteelle tehtävä päivityksen tekeminen on ajankohtainen ja hyödyllinen aihe. Tavoitteena tässä insinöörityössä oli tehdä oma Linux-käyttöjärjestelmä sulautetulle laitteelle, joka voidaan päivittää etänä. Oman Linux-käyttöjärjestelmän suunnitteleminen ja toteuttaminen on todella työläs prosessi, siksi on luontevaa käyttää työkalua, jonka avulla voidaan keskittyä ominaisuuksien kehittämiseen ja käyttää olemassa olevia ratkaisuja.

2 MENDER

Mender on avoimen lähdekoodin alusta, jolla pystytään päivittämään etänä sulautettu Linux-pohjainen laite. Laitteiston päivittäminen etänä on suosittu tapa, koska tällöin voidaan päivitys tehdä kokonaan etänä Internetin välityksellä, jolloin ei tarvitse olla kontaktissa laitteeseen fyysisesti. [2]

Usein laitevalmistajat ovat käyttäneet omaa ratkaisua laitteen etäpäivittämiseen. Oman ratkaisun käyttäminen on johtanut kuitenkin siihen, että laitteiden päivittämisestä on tullut hankala, turvaton ja epäonnistumisen riski kasvaa. Epäonnistuminen on mahdollista, esim. Internet yhteyden tai virran katkettua. Menderissä on näihin skenaarioihin ratkaisu, joista kerrotaan myöhemmissä luvuissa. [3]

Päivitysohjelmiston kehittäminen on yleensä ollut sulautettujen järjestelmien ohjelmistokehittäjien mukavuusalueen ja osaamisen ulkopuolella. Monesti kehittäjät ovat tuoneet päivitysmekanismin vasta sen jälkeen, kun tuote on tullut markkinoille ja asiakkaat ovat alkaneet vaatimaan päivityksiä. Päivitysmekanismin toteuttaminen jälkikäteen nopealla aikataululla, lisää riskiä, jolloin luotettavuus ja turvallisuus kärsii. [3]

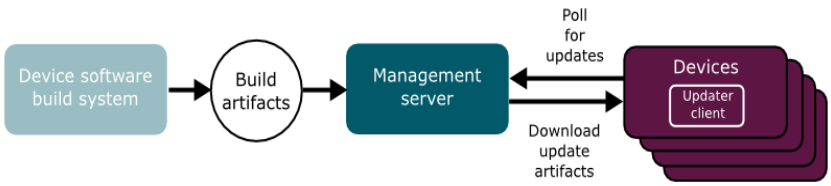
Mender toimittaa luotettavan, turvallisen ja testatun järjestelmän, joka on avoin kaikille. Kehittäjät voivat keskittyä enemmän ominaisuuksien kehittämiseen laitteessa, kun päivitys mekanismin miettimiseen. Käyttöjärjestelmän kehityksessä ollaan tukeudettu pitkään avoimen lähdekoodin projekteihin, kuten Linux ja FreeRTOS, jolloin on luonnollista, että etäpäivittämisessä voidaan hyödyntää avointa lähdekoodia, koska usein eri päivitysohjelmistojen perusasiat ovat kaikilla hyvin samanlaiset. Tehdessäni tätä työtä Mender-tuki oli saatavilla Yoctolla koottuun Linux-jakeluversioon ja binäärimuodossa olevaan Debian-jakeluversioon. Lisäksi Mender-dokumentit ilmaisivat, että on myös mahdollista lisätä Mender muillekin alustoille. [3]

2.1 Ominaisuudet

Menderissä on monia hyödyllisiä ominaisuuksia, kuten mahdollisuus muodostaa laitteista ryhmiä, joka mahdollistaa, että voidaan päivittää kaikki ryhmään kuuluvat laitteet yhdellä kerralla, jolloin ei tarvitse päivittää jokaista laitetta yksi kerrallaan.

Mender tarjoaa graafisen käyttöliittymän laitteiden hallinnointiin ja laitteet voidaan jakaa esim. laitteessa olevan ohjelmistoversion tai laitteen tyypin mukaan. Menderin graafinen käyttöliittymästä löytyy erittäin hyödyllinen inventaario (engl. inventory), jonka avulla laitteesta on saatavilla erilaista tietoa esim. IP-osoite, viimeaikaisin yhteys palvelimille ja Linux-ydin versio. Edellä mainittuun inventaarion on mahdollista myös lisätä omia attribuutteja, ja tätä toiminnallisuutta on myös tarkoitus hyödyntää tässä työssä. Lisäksi Menderin graafisessa käyttöliittymässä on loki työkalu, jolla pystyy selvittämään ongelmia, joita ilmentynyt päivityksen epäonnistumisen yhteydessä.

2.2 Toimintaperiaate



Kuva 25. Menderin toimintaperiaate [13]

Menderin toimintaperiaate on mielestäni yksinkertainen, mutta taustalla on asioita, jotka tekevät siitä luotettavan ja turvallisen. Yksinkertaisen siitä tekee se, että siinä on jaoteltu selkeästi sen rakenne, eli miten se toimii. Rakenne voidaan jakaa neljään eri komponenttiin.

Koontijärjestelmä, yleensä PC, jolla kootaan kohdelaitteelle käyttöjärjestelmä. Tässä työssä PC on pöytäkone, jossa on Ubuntu 18.04 -käyttöjärjestelmä, kohdelaite on Raspberry Pi 3, jonka käyttöjärjestelmänä on Yocto-työkalulla koottu Linux-järjestelmä. [4]

Koontiartefaktin, joka on käytännössä tekstitiedosto, joka sisältää päivitykseen liittyvät muutokset. Artefakti on siis se, joka lähetetään laitteelle, joka halutaan päivittää. Artefaktit sisältävät seuraavan metatiedot: nimen, jossa on ohjelmiston versio, kohdelaitteen tyyppi ja tarkastussumman. [4]

Hallinnointipalvelin, jonka avulla laitteille voidaan lähettää uusia päivityksiä. Hallinnointi palvelimeen kuuluu graafinen käyttöliittymä. Palvelimen käyttöliittymän kautta näkee kaikki laitteet, laitteiden yhteyden tila tai aika jolloin ollaan muodostuttu yhteys palvelimeen. Käyttöliittymän kautta tehdään myös päivitykset. [4]

Kohdelaite, jossa on asiakasohjelma, joka tässä työssä on Raspberry Pi 3. Käytännössä kohdelaitteet ovat yhteydessä palvelimeen, sille määritetyn aikavälein ja pyytävät tietoa päivitysten saatavuudesta. [4]

2.3 Päivitysprosessi

Menderissä on panostettu siihen, että laitteen päivittäminen on turvallista ja luotettavaa. Laitteen päivittämisen pitää olla luotettava, koska sen pitää olla valmis tilanteisiin, joita on vaikea ennustaa esim. verkkoyhteyden tai virran katkeaminen. Menderissä on käytössä ylimäärävarmistus-mekanismi (dual redundant scheme), joka toimii käytännössä siten, että käytössä on kaksi levyosioita, osio A ja osio B. Toinen levyosio on edellinen käytössä ollut levyosio, ja se ollaan merkattu lipulla epäaktiivinen, jotta käynnistyksenlataaja ei lataa sitä. Käytössä oleva osio on merkattu lipulla aktiivinen, jonka käynnistyksenlataaja lataa käynnistyksen yhteydessä. Uusi päivitys lähdetään aina kirjoittamaan siihen osioon, jonka lippu on epäaktiivinen. Päivityksen onnistuttua epäaktiivinen levyosio muutetaan aktiiviseksi, jolloin toinen levyosio, joka on ollut aktiivinen, muutetaan epäaktiiviseksi. Levyosioiden muutos tulee voimaan seuraavalla uudelleenkäynnistyksellä. [4]

Mikäli uudelleenkäynnistymisen yhteydessä todetaan, että päivitys onnistui, kohdelaite tekee vahvistuksen (engl. commit) päivityksestä. Vahvistuksen tarkoitus on kertoa palvelimelle, että käyttöjärjestelmä on käynnistetty oikein. Mikäli käyttöjärjestelmä käynnistyy ennen, kuin vahvistus ollaan suoritettu, tietää järjestelmä, että jokin on mennyt vikaan siitäkin huolimatta, että päivitys onnistui. Tällöin mennään takaisin siihen osioon, joka tässä tapauksessa on uusi epäaktiivinen, johon ei oltu vielä kopioitu uutta aktiivista osiota. [4]

Päivitys tarkoittaa Menderissä sitä, että käyttöjärjestelmän käytössä oleva juuritiedostojärjestelmä korvataan uudella, mikä tarkoittaa sitä, että kaikki juuritiedostojärjestelmässä olevat tiedostot korvataan uusilla. Tämän takia kaikki konfiguraatiot pitää olla sijoitettu toiselle levyosiolle. Levyosiot kannattaa toteuttaa Menderin ohjeiden mukaan, jotta päivittäminen olisi mahdollisimman luotettavaa. Vaatimuksen alarajana on neljä eri levyosiota. Yksi osio, joka sisältää käynnistyksenlataajan ja sen ympäristön, toinen osio, jossa on juuritiedostojärjestelmä ja Linux-ydin. Kolmas levyosio on varattu niille tiedostoille, jotka sisältävät ne tiedostot mitä ei korvata uusilla päivityksen yhteydessä eli tähän

levyosioon voidaan tallentaa dataa. Neljäs levyosio on kopio siitä levyosiosta, joka sisältää juuritiedostojärjestelmän ja Linux-ytimen.[5]

3 YOCTO-PROJEKTI

Yocto-projekti on Linux-säätiön hallinnoiva avoimen lähdekoodin projekti, jonka tehtävänä on antaa kehittäjille työkalut, joilla voidaan rakentaa arkkitehtuurista riippumatta, ennen kaikkea sulautettuihin järjestelmiin kustomoitu Linux levykuva. Yocto-projekti on käytännössä OpenEmbeddedin ja Pokyn yhdistelmä. Työkalut joita käsitellään seuraavaksi ovat: OpenEmbedded-Core, Poky ja BitBake. [6]

3.1 Historia

Yocto Project sisältää paljon erilaisia termejä ja konsepteja kuten esim. OpenEmbedded, OpenEmbedded-Core, Poky, BitBake, kerros (engl. layer) ja reseptit (engl. recipes). Näiden ymmärtäminen on tärkeää jatkon kannalta, mutta kun aloittaa selvittämään mitä nämä tarkoittavat huomaa nopeasti, että termit lentävät paikasta toiseen ja lopputuloksena on se, että on vaikea ymmärtää kokonaiskuvaa. Näin ainakin itselleni kävi, joten päätin lähteä liikkeelle tutkimalla, miten kaikki alkoi, ja tutustuin Yocto-projektin historiaan. Tutustuessani Yocto-projektin historiaan alkoi asiat loksahtamaan paikoilleen. Tästä syystä päätin sisällyttää Yocto-projektin liittyvän historian tähän opinnäytetyöhön, lyhyesti. [7]

Yocto-projektin juuret ovat peräisin OpenEmbedded projektista, johon liittyi useita projekteja, joiden tarkoitus oli portata Linux muutamiin kämmentietokoneisiin (engl. Pocket PC), esim. HP Compaq iPaq, jossa oli Pocket PC 2000:sta Windows Mobile 6.5 käyttöjärjestelmään. Nykyisin edellä mainittu viritys löytyy kerroksena nimeltä meta-handheld. OpenEmbededed tuli eloon v. 2003 koontijärjestelmänä kämmentietokoneille, mutta myöhemmin se laajensi kämmentietokoneista muihin sulautettuihin alustoihin. Tarkoituksena oli myös mahdollistaa binääripakettien asentaminen käyttäen ipk formaattia. Asentaminen oli mahdollista tekemällä reseptejä jokaiselle ohjelmalle ja käyttämällä BitBake työkalua tehtävien aikatauluttajana (engl. task scheduler). Tästä seurasi se, että käyttämällä haluttuja reseptejä voitiin luoda Linux jakeluversion, jossa on käyttäjän määrittelemät vaatimukset. [7]

Vuonna 2005 Richard Purdie, joka oli tuolloin kehittäjänä yrityksessä nimeltä OpenedHand teki uuden haaran OpenEmbeddedestä, jonka hän nimesi Pokyksi. OpenedHand

kehitti Pokystä älypuhelimelle ja kämmentietokoneille suunnatun Linux-pohjaisen käyttöjärjestelmän. [7]

Molemmat, Poky ja OpenEmbedded jatkoivat kehitystä jakaen toisilleen päivityksiä ja koodia. Intel osti OpenedHandin v. 2008 ja v. 2010 Intel siirsi Pokyn Linux-säätiölle, jolloin projektin nimeksi tuli Yocto-projekti. [7]

Vuodesta 2010 lähtien yleiset komponentit, jotka OpenEmbedded ja Poky jakoivat, yhdistettiin erilliseksi projektiksi nimeltä OpenEmbedded-Core, jota myös referoidaan oe-core nimellä. Nykyään Yocto-projektin ydin on siis Poky, josta tuli referenssi jakeluversio, OpenEmbedded-Core, joka sisältää ydin metadatan eli keskeisimmät reseptit ja BitBake, joka toimii tehtävien aikatauluttajana. [7]

3.2 Reseptit

Reseptit ovat tiedostoja jotka sisältävät tiedon siitä mm. missä kukin ohjelma sijaitsee, mitä sille pitää tehdä ja miten se käännetään lähdekoodista. Esimerkiksi reseptiä voi verrata GNU make-tiedostoon. Reseptit noudattavat tiettyä syntaksia, jotta BitBake osaa tulkita ne oikein. Erityisesti polkujen nimeäminen tuotti aluksi vaikeuksia ymmärtää niitä, mutta löytäessäni Yocto-projektin lähdekoodista src/poky/meta/conf/bitbake.conf tiedoston jossa oltiin selitetty nimeäminen, joten se ei tuottanut sen jälkeen vaikeuksia. [8]

```
do_install() {
    install -d ${D}${sbindir}
    install -m 0755 watchdogdeamon ${D}${sbindir}


}
```

Kuva 1. esim. Yocto-projektin omista polkumuuttujista.

Esim. Binääritiedosto wathcdogdeamon asennetaan sbindir hakemistoon, joka kääntyy siten, että sbindir tarkoittaa "${exec_prefix}/sbin." Sbin on tuttu hakemisto Linuxissa ja exec_prefix kääntyy "${prefix}" nimiseksi, mikä ei ainakaan itselläni sanonut yhtään mitään, mutta konfiguraation mukaan tuo kääntyy "/usr", joka on tuttu hakemisto Linuxista.

```
export sbindir = "${exec_prefix}/sbin"
export prefix = "/usr"
export exec_prefix = "${prefix}"
```

Kuva 2. Toinen esim. Yocto-projektin omista polkumuuttujista.

Eli lopuksi saadaan ulos /usr/sbin hakemisto, minne binääritiedosta asennetaan.

3.3 Kerros

Kerros ovat kansio, joka sisältää reseptejä ja kerroksen konfiguraatio tiedoston. Kerros voi myös sisältää edelliseen konfiguraatioon olevia muutoksia ja asetuksia. Kerroksia voidaan suositella käytättävän siten, että koko projektia ei tulisi laittaa yhteen kerrokseen. Käyttämällä useaa kerrosta eri toiminnallisuuksille, mahdollistaa kerroksen uudelleenkäyttämisen ja korjaamisen. [8]

3.4 OpenEmbedded-Core-ydinkerros

OpenEmbedded-Core sisältää perustan resepteille, luokille ja muille tiedostoille joiden on tarkoitus olla jaettavissa kaikille OpenEmbedded muunnoksille kuten Yocto Project. OpenEmbedded-Core -yhteisprojekti, OpenEmbeddedin ja Yocton välillä mahdollistaa paremmin kontrolloidun ja laadukkaamman lopputuloksen. [6,8]

3.5 Poky-referenssi -jakeluversio

Poky on Yocto-projektin referenssi jakeluversio, joka sisältää OpenEmbededed koontijärjestelmän ja metatiedot, joilla pääsee alkuun luodessa omaa jakeluversiota. Poky sisältää kaikki Yocto Projectiin liittyvät dokumentit, lisenssitiedot, BitBake komennon skriptit ja useita eri reseptejä. Poky ei sisällä binääritiedostoja, sen sijaan lähdekoodi tarvitsee koota itse. Kuuden kuukauden välein julkaistaan uusi versio. Huolimatta siitä, että Poky on toimiva ja testattu jakeluversio, niin sitä ei suositella suoraan käytettäväksi sen nykyisessä muodossa. [8]

3.6 BitBake-koontityökalu

BitBake on koontityökalu, joka mahdollistaa ristiin kääntämisen, yleensä sulautettuihin Linux-järjestelmissä. BitBake on nykyään itsenäinen projekti, mutta se kuului aikoinaan OpenEmbededed-projektiin. BitBaken tehtävänä on jäsentää metadataa, generoida tehtävälista jäsennetyistä metadatasta ja suorittaa tehtävälistan tehtävät. BitBake sisältää monia komentorivi argumentteja, jotka voivat tuoda helpotusta koonti prosessiin. [8,9]

3.7 Yocto-projektin hyödyt

Käyttöjärjestelmän kehittäminen tyhjästä on erittäin vaikea, aikaa vievää ja kallis projekti. Yocto-projektin avulla voidaan luoda kustomoitu Linux-käyttöjärjestelmä, joka sisältää ne komponentit, joita kehittäjä haluaa tai tarvitsee. Useat puolijohdevalmistajat ovat tehneet omille tuotteilleen oman BSP(Board support package) -paketin, joka tarjoaa valmiit reseptit raudan tukemiseen. Mikäli valmista BSP pakettia ei löydy, niin Yocto-projektissa on ohjeet, miten voi tehdä oman BSP -paketin. [6,10]

Yocto-projektin tuotos on helposti siirrettävissä toiselle arkkitehtuurille, tämä tarkoittaa sitä, että arkkitehtuurin vaihtuessa voidaan käyttää olemassa olevaa koodia toisella arkkitehtuurilla. [10]

Yocto-projekti tarjoaa referenssi jakeluversion, jolla kehittäjät pääsevät nopeasti alkuun. Referenssi jakeluversio sisältää vain tarvittavat komponentit, jotta sen muistijalanjälki olisi mahdollisimman pieni. Komponentit ovat kuitenkin helposti vaihdettavissa, mikäli kehittäjä haluaa tehdä niihin muutoksia. [6]

Yksi tärkein Yocto-projektin ominaisuuksia on se, että sen versioita testataan ja arvioidaan jatkuvasti. Tämä on mahdollista, koska Yocto-projektin ekosysteemi on avoin ja yhteisöön kuuluu niin yksilöitä kuin organisaatioita.[6]

4 TYÖN SUORITTAMINEN

4.1 Mender palvelimen asentaminen

4.1.1 Tavoite

Tavoitteena on asentaa Mender-palvelin PC:lle, johon kohdelaite voi ottaa yhteyden, jotta voidaan mahdollistaa kohdelaite etäpäivitys ja toiminnan seuranta. Lisäksi tavoitteena on myös opetella Menderin käyttöä.

4.1.2 Suunnittelu

Palvelimen asentaminen vaatii, että käytössä on PC, josta löytyy riittävästi tallennustilaa ja muistia. PC:n käyttöjärjestelmäksi valitaan jokin Linux-jakeluversio, koska Menderin dokumentit suosittelevat Linuxin käyttöä. Lisäksi staattinen IP-osoite on hyvä olla, koska palvelimen IP-osoite kova koodataan Yocton koontiin, jolloin ei tarvitse huolehtia vaihtuvaa IP-osoitetta. Virtuaalikone mahdollistaa helpon tavan kokeilla asioita, joista ei ole aiempaa kokemusta, koska siinä mahdollista tehdä tilannevedoksia (engl. snapshot), tämän seurauksena palvelin asennetaan virtuaalikoneeseen.

4.1.3 Suoritus

Mender-palvelin kannattaa asentaa Menderin virallisten dokumenttien mukaan, koska ne tarjoavat ajantasaisimmat ohjeet palvelimen asentamiseen. Menderistä on saatavilla joitain eri ohjelmistoversioita, mutta tässä työssä käytettiin uusinta versiota, suorituksen tekohetkellä 1.7, joka oli saatavilla. Mender voidaan asentaa käyttämällä testiversiota, joka toimii paikallisesti, koska tämä on ilmainen ja helppo tapa kokeilla Menderiä. Testiversio ei kuitenkaan sovellu tuotantotasolle, jolloin on hyvä asentaa tuotantotasolle sopiva palvelin. Lisäksi Mender tarjoaa maksullista palvelua nimeltä Hosted Mender, joka toimii käytännössä avaimet käteen -periaatteella.

Testiversion ja tuotantoversion välillä käytiin pohdintaa ja molempia testattiin, mutta lopuksi tässä työssä päädyttiin käyttämään testiversiota, koska sen käyttämiseen ei

liittynyt mitään kustannuksia toisin, kun tuotantoversioon, jossa olisi tarvittu julkinen IP-osoite ja verkkotunnus. Reitittämiseen liittyviä kokeiluja ei ollut mielenkiintoa tehdä omassa verkossa, joten tuotantopalvelinta kokeiltiin Amazonin EC2 -ympäristössä ja IP-osoitteiden ja verkkotunnuksen reititys tehtiin Amazonin Route53-palvelussa. Kohdelaitetta muodostamissa epäonnistuttiin, eikä yhteyttä palvelimeen ja tuntihintaista EC2-ilmentymää (instance) ollut mielekästä pitää, joten tästä syystä tuotantopalvelimesta luovuttiin.

Virtuaalikoneeksi valittiin Virtualbox, koska se on ilmainen ja yleisessä käytössä oleva ohjelmisto. Käyttöjärjestelmäksi valittiin Ubuntu 18.04, johon määriteltiin 3096 MB muistia, 30 GB tallennustilaa, kaksi prosessoriydintä ja verkoksi valittiin sillattu adapteri, jotta asiakasohjelma löytäisi lokaalissa verkossa olevan palvelimen. Testipalvelimen vähimmäisvaatimukset olivat dokumenttien mukaan 10 GB tallennustilaa ja kaksi GB muistia, mutta virtuaalikoneen isäntäkone oli pöytäkone, jossa oli riittävästi resursseja, jolloin Mender-palvelimelle annettiin reilusti resursseja käyttöön. Virtuaalikoneen asennuksen jälkeen voitiin aloittaa Mender palvelimen asentaminen virtuaalikoneelle.

Mender-palvelin on suunniteltu mikropalveluarkkitehtuurin (engl. microservices) mukaan, joka tarkoittaa sitä, että se koostuu useista pienistä erillään olevista palveluista. Eri palvelut ollaan koottu yhteen ja yhdistetty toisiinsa Docker Compose työkalun avulla. Tästä syystä Virtuaalikoneelle tarvitsee asentaa Docker Compose. Ennen kuin Docker Compose voidaan asentaa, tarvitsee asentaa Docker Engine, jota Docker Compose tarvitsee. Docker Engine:n asentaminen Ubuntulle tapahtuu siten, että ensin päivitetään apt paketti indeksi ajamalla seuraava komento terminaalissa:

```
- $ sudo apt-get update
```

Tämän jälkeen asennetaan tarvittavat paketit, jotta voidaan asentaa paketteja HTTPS:n kautta:

```
- $ sudo apt-get install \
   apt-transport-https \
   ca-certificates \
   curl \
   gnupg-agent \
   software-properties-common
```

Lisätään Dockerin virallinen GPG-avain:

```
- $ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add –
```

Curl-työkalulla ladataan avain URL:sta ja apt-key add -komennolla lisätään avain käyttöjärjestelmään.

Varmistetaan, että oikea avain tuli lisättyä siten, että verrataan Dockerin sivulla kerrottuun sormenjälkeen, joka on 9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88.

Vertaaminen voidaan toteuttaa siten, että vertaamalla Dockerin sivuilla lukeneesta sormenjäljestä otetaan viimeiset kahdeksan merkkiä ja katsotaan omalta koneelta, mitä tietoja kyseinen sormenjälki sisältää:

```
- $ sudo apt-key fingerprint 0EBFCD88
```

Mikäli komennon output on seuraava, voidaan todeta, että oikea avain tuli lisättyä:

```
pub   rsa4096 2017-02-22 [SCEA]
      9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88
uid   [ unknown] Docker Release (CE deb) <docker@docker.com>
sub   rsa4096 2017-02-22 [S]
```

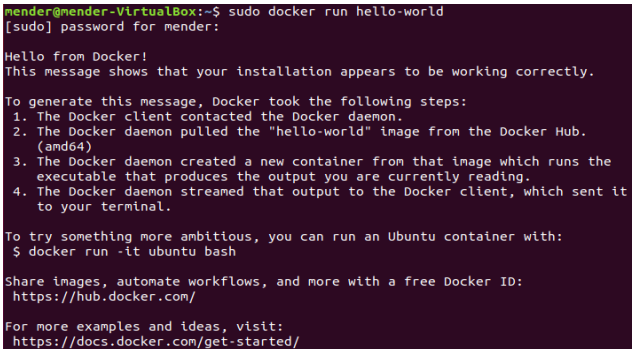
Lisätään Dockerin vakaa repositori:

```
- $ sudo add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
   $(lsb_release -cs) \
   stable"
```

Tämän jälkeen voidaan asentaa itse Docker Engine:n paketti, seuraavalla komenolla:

```
- $ sudo apt-get update
- $ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

Ajamalla kuvan kolme mukaan Dockerin hello-world -kuvake voidaan todistaa, että asennus meni oikein.



Kuva 3. Onnistunut Docker CE asennus

Docker Compose voidaan asentaa siten, että seuraavalla komennolla ladataan vakaa julkaisu:

```
- $ sudo curl -L "https://github.com/docker/compose/releases/download/1.24.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

Ladatut tiedostot ohjataan o-argumentilla, argumentin jälkeiseen polkuun. Tämän jälkeen annetaan suoritus oikeudet binääritiedostolle:

```
- $ sudo chmod +x /usr/local/bin/docker-compose
```

Testataan, että asennus meni oikein ajamalla seuraava komento jolla nähdään docker-compose:n versio:

```
- $ docker-compose –version
```

```
mender@mender-VirtualBox:~$ docker-compose --version
docker-compose version 1.23.2, build 1110ad01
```

Kuva 4. Onnistunut Docker Compose asennus

4.1.4 Menderin asennus

Asennus tapahtuu siten, että kloonataan git-työkalulla Menderin Github-sivulta lähdekoodi. Kloonaaminen tehdä terminaalissa ajamalla git clone, argumentiksi voidaan esimerkiksi valita jokin tietty haara versioista, eli tässä työssä käytetty 1.7 versio. Tiedostot tallennetaan integration-1.7.0 kansioon.

```
- $ git clone -b 1.7.0 https://github.com/mendersoftware/integration.git integration-1.7.0
```

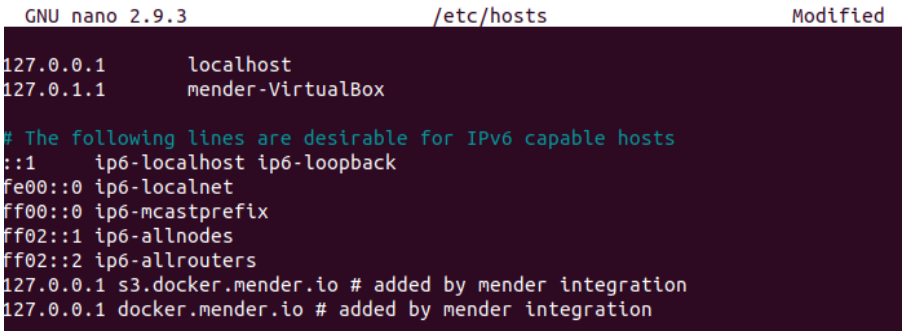
Kloonauksessa voi mennä pieni hetki, mikäli käytössä ei ole nopeaa verkkoyhteyttä. Kloonauksen jälkeen voidaan navigoida cd-komennolla sinne kansioon, minne tiedostot tallennettiin.

```
- $ cd integration-1.7.0
```

Ennen kuin palvelin voidaan käynnistää, voidaan lisätä kaksi merkintää käyttöjärjestelmän host-tiedostoon.

Avataan /etc/hosts-tiedosto nano-editorilla ja lisätään seuraavat merkinnät:

```
127.0.0.1 s3.docker.mender.io ja 127.0.0.1 docker.mender.io
```



Kuva 5. Nano-editorilla muokataan hosts-tiedostoa.

Tämä ei kuitenkaan ole välttämätöntä tehdä, sillä ensimmäisen käynnistyksen yhteydessä lisäykset lisätään automaattisesti skriptillä, joka käynnistää palvelimen.

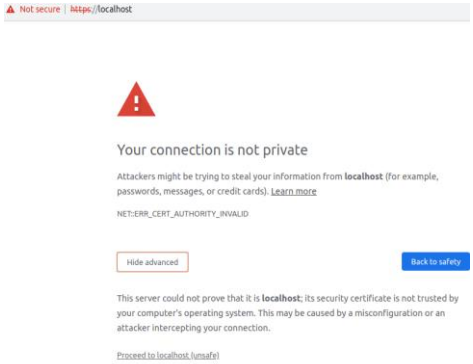
Palvelin käynnistetään ajamalla up-skripti:

```
- $ ./up
```

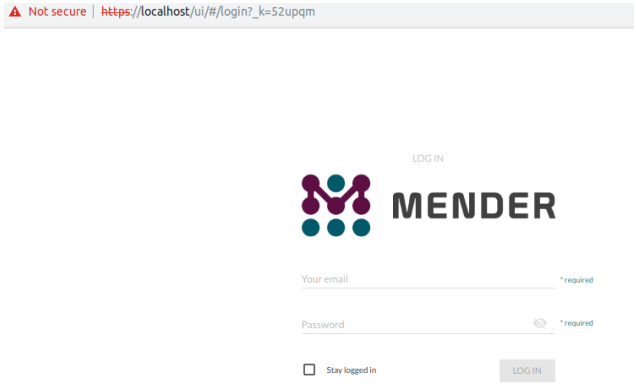
Käyttäjätunnuksen lisääminen tehdään siten, että palvelin on oltava päällä, joten avataan uusi terminaali ikkuna ja navigoidaan cd-komennolla integration-1.7.0 kansioon ja ajetaan seuraava skripti.

```
- $ sudo ./demo exec mender-useradm /usr/bin/useradm create-user --
username=myusername@example.com --password=mysecretpassword
```

Mender käyttöliittymä löytyy nyt https://localhost/ osoitteesta. Mender:n yhdyskäytävä sallii vain salatun kommunikoinnin TLS salausprotokollaa käyttäen. Tämän takia selaimen tarvitsee hyväksyä yhdyskäytävän sertifikaatti.



Kuva 6. Selaimessa navigoidaan localhostiin.



Kuva 7. Kirjautuminen Mender-palvelimen käyttöliittymässä.

4.1.5 Staattisen IP-osoitteen määrittäminen

On myös hyödyllistä laittaa palvelimeen staattinen IP-osoite, jotta kohdelaite löytää sen, koska palvelin pyörii virtuaalikoneella, niin se tullaan aina silloin tällöin sammuttamaan, joka voi johtaa IP-osoitteen muuttumiseen. Staattinen IP-osoite voidaan määrittää Ubuntussa, käyttämällä netplan-työkalua siten, että ensin generoidaan uusi netplan-konfiguraatio terminaalissa, seuraavasti:

```
-$ sudo netplan generate
```

Tämän jälkeen voidaan lisätä seuraava konfiguraatio, jolla saavutetaan staattinen IP-osoite, avaamalla tiedosto tekstieditoriin seuraavasti:

```
-$ sudo nano /etc/netplan/01-network-manager-all.yaml
```

Lisätään uusi konfiguraatio:

```
network:
 version: 2
 renderer: NetworkManager
 ethernets:
  enp0s3:
   dhcp4: no
   dhcp6: no
   addresses: [192.168.1.199/24, ]
   gateway4:  192.168.1.1
   nameservers:
        addresses: [8.8.8.8, 8.8.4.4]
```

Konfiguraatio tarvitsee olla tietyn syntaksin mukainen ja tabulaattoria ei voi käyttää, tämän takia pitää laittaa tyhjiä välilyöntejä niihin kohtiin, joissa niitä tarvitaan. Tämän jälkeen tallennetaan konfiguraatio ja otetaan muutokset voimaan netplan apply -komennolla.

```
-$ sudo netplan apply
```

4.2 Yocto-kehitysympäristön asentaminen

4.2.1 Tavoite

Tavoitteena on asentaa PC:lle Yocto-projektin kehitysympäristö, jolla voidaan koota Raspberry Pi 3:lle levykuva. Lisäksi tavoitteena on myös oman kerroksen tekeminen sekä Menderin kerroksen lisääminen, jotta kohdelaite voidaan yhdistää Mender-palvelimineen. Tarkoitus on myös todentaa yhteyden toimiminen ja tehdä päivitys kohdelaitteeseen Mender-palvelimen avulla.

4.2.2 Suunnittelu

Yocto-projektin kehitysympäristö asennettiin Ubuntu 18.04 -käyttöjärjestelmälle. Käytössä oli pöytäkone jossa oli neliydinprosessori, 8 GB muistia ja siihen oltiin varattu tallennustilaa n. 150 GB Yocton output-tiedostoja varten, joka oli enemmän kuin riittävästi, sillä Yocton suositus on, että tallennustilaa varattaisiin 50 GB per koonti.

Valitaan uusin saatavilla oleva Yocto-julkaisu, joka oli teko hetkellä 2.6 Thud. Lisäksi seuraavaksi luetelluista ohjelmistoista vähintään: git v.1.8.3.1, tar v.1.27 ja Python v.3.4.0 tulisi löytyä kehitysympäristöstä. Mikäli käytöstä löytyy uusin LTS versio esim. Ubuntusta, eli tällä hetkellä kirjoittaessa 18.04, niin todennäköisesti sopiva git, tar ja Python versio löytyy jo valmiiksi asennettuna.

PC:n lisäksi tarvitaan Raspberry Pi 3, sd-muistikortti, virtalähde ja Ethernet-kaapeli. Tarkoitus on kirjoittaa Yocton koonnista tuleva levykuvake ja sen jälkeen kaikki muutokset käännökseen suoritetaan päivityksenä Mender-palvelimen kautta. Ennen Yocto-koonnin tekemistä, tarvitsee tunnistaa mitä kerroksia tarvitaan. Raspberry Pi:lle on saataville oma BSP, joka on kokoelma tietoa siitä mitä rautaa on käytössä. Mender-yhteys saadaan toimimaan Mender-kerroksen avulla.

Yocto-koonnin tekemiseen liittyy paljon konfigurointia ja työ on hyvin kokeiluluontoista, jonka takia käytetään git-ohjelmistoa versionhallintaan.

4.2.3 Suoritus

Yocto-projektilla on joitain vaadittuja paketteja, jotka tarvitsevat olla PC:llä, jolla suoritetaan koonti, joten asennetaan ensin kaikki tarvittavat paketit seuraavasti:

1. Päivitetään lista paketeista, jotta saadaan asennettua uusimmat versiot paketeista joita tarvitaan.

Terminaalissa ajetaan komento:

```
- $ sudo apt update
```

2. Tämän jälkeen asennetaan kaikki tarvittavat paketit.

Terminaalissa ajetaan seuraava komento:

```
- $ sudo apt-get install -y gawk wget git-core diffstat unzip texinfo gcc-multilib \

build-essential chrpath socat cpio python python3 python3-pip python3-pexpect \

xz-utils debianutils iputils-ping libsdl1.2-dev xterm
```

Lähdekoodit ladataan kansioon nimeltä Yocto, johon tehdään alikansio nimeltä src, minne ladataan kaikki lähdekoodit. Tässä työssä kaikki tarvittavat lähdekoodit saadaan kloonattua Githubista. Kloonaus tapahtuu ajamalla git clone -komento terminaalissa ja lisäämällä haluttu haara ja repositorin URL argumentiksi.

```
- $ git clone -b thud git://git.yoctoproject.org/poky

- $ git clone -b thud git://git.openembedded.org/meta-openembedded

- $ git clone -b thud git://git.yoctoproject.org/meta-raspberrypi

- $ git clone -b thud git://github.com/mendersoftware/meta-mender
```

Kloonauksen jälkeen tehdään kansio, joka nimetään meta-wathcdogservice, josta tulee työssä oleva oma kerros. Yocto-projektin dokumenteissa ohjataan nimeämään kerros siten, että ensin tulee meta-jakerroksennimi.

```
random@Yocto:~/Yocto/src$ ls
meta-mender  meta-openembedded  meta-raspberrypi  meta-watchdogservice  poky
```

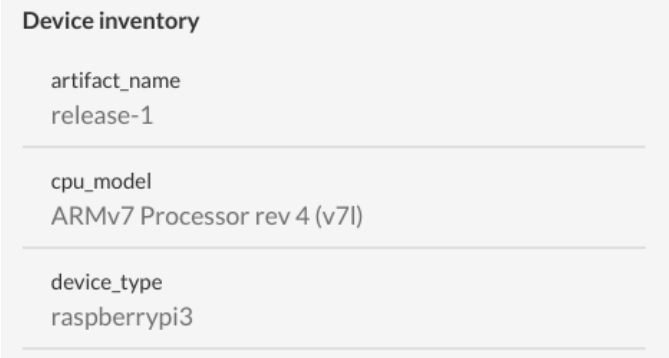
Kuva 8. Kloonatut kerrokset ja oma kerros.

Tarkoitus on tehdä oma kerros, joka sisältää reseptin, joka asentaa skriptin, jonka avulla nähdään Mender-palvelimella Raspberry Pi:n käyttöaika, ja skripti jonka avulla nähdään uusin epäonnistunut ssh-kirjautumisyritys. Kerros koostuu kahdesta kansiosta conf ja recipes-watchdogservice. Conf-kansio sisältää konfigurointitiedoston nimeltä layer.conf, joka nimetään samalla tavalla kaikissa kerroksissa. Layer.conf-tiedoston asetukset:

```
BBPATH .= ":${LAYERDIR}"

BBFILES += "${LAYERDIR}/recipes-*/*/*.bb \

          ${LAYERDIR}/recipes-*/*/*.bbappend"


BBFILE_COLLECTIONS += "watchdogservice"

BBFILE_PATTERN_watchdogservice ="^${LAYERDIR}/"

BBFILE_PRIORITY_watchdogservice = "9"

LAYERSERIES_COMPAT_watchdogservice = "thud"

LAYERVERSION_watchdogservice = "1"

LICENSE_PATH += "\${COMMON_LICENSE_DIR}"
```

Recipes-watchdogservice-kansio sisältää menderinventory nimisen kansion, jossa on files niminen kansio, joka sisältää skriptit jotka halutaan lisätä käännökseen sekä menderinventory.bb nimisen tiedoston, joka on resepti, joka sisältää tiedot siitä, miten skriptit asennetaan.

Menderissä on olemassa käsite inventaario, jonka avulla voidaan näyttää informaatiota kohdelaitteesta Mender-palvelimessa. Kuva 9 näyttää esimerkin miltä inventaario näyttää Mender-palvelimen käyttöliittymässä. Tavoitteena on luoda inventaarioon failed_ssh_login-atribuutti, joka näyttää viimeisimmän epäonnistuneen SSH-kirjautumisyrityksen ja uptime-atribuutti, joka näyttää kohdelaitteen käynnissä olleen käyttöajan.



Kuva 9. Menderissa oleva inventaario.

Inventaarion tekemiseen tehdään kaksi bash-skriptiä, joille annetaan nimeksi mender-inventory-failedssh.sh ja mender-inventory-uptime.sh. Inventaarion skriptin nimi tarvitsee noudattaa tiettyä syntaksia, mender-inventory-jokinomanimi, jollei syntaksia ei ole noudatettu niin sitä ei huomioida, jolloin sitä ei näy inventaarissa.

Skriptit:

```
#!/bin/sh

#mender-inventory-failedssh.sh

# Failed ssh login

#Error testaus, että lopetaan suorittaminen virheen tullessa

set -ue

#Asetaan default kieli outputille

LC_ALL=C

export LC_ALL

#failed_ssh on attribuutin nimi ja

#/var/log/auth.log näyttää kirjautumis lokit ja sieltä etsitään kohta Failed password

#ja tail komennolla saadaan listan viimeinen kohta joka on uusin tapaus

echo "failed_ssh=$(cat /var/log/auth.log | grep "Failed password" | tail -n 1)"
```

```
#!/bin/sh

# mender-inventory-uptime.sh

# Get Uptime

#Lopetaan suorittaminen, mikäli tulee error

set -ue

#Asetaan default kieli outputille

LC_ALL=C

export LC_ALL

#uptime on attribuutin nimi ja "$(cat /proc/uptime)" saadaan uptime

echo "uptime=$(cat /proc/uptime)"
```

Seuraavaksi tarvitsee tehdä resepti, joka kuvaa miten, tässä tapauksessa bash-skripti asennetaan ja minne polkuun. Reseptin kirjoittaminen tehdään siten, että tehdään ensin tiedosto menderinventory.bb joka näyttää seuraavalta:

```
SUMMARY = "Mender inventory scripts summary"

LIC_FILES_CHKSU = "file://${COMMON_LICENSE_DIR}/MIT;md5=0835ade698e0bcf8506ecda2f7b4f302"

DESCRIPTION = "Mender inventory scripts"

LICENSE = "MIT"

SRC_URI = "file://mender-inventory-uptime.sh \

          file://mender-inventory-failedssh.sh \

          "

S = "${WORKDIR}"

do_install() {

install -d ${D}${datadir_native}/mender/inventory/

install -m 0755 ${WORKDIR}/mender-inventory-uptime.sh ${D}${datadir_native}/mender/inven-
tory

install -d ${D}${datadir_native}/mender/inventory/

install -m 0755 ${WORKDIR}/mender-inventory-failedssh.sh ${D}${datadir_native}/mender/in-
ventory

}FILES_${PN} = "${datadir_native}/mender/inventory"
```

SUMMARY- ja DESCRIPTION-muuttajat ovat nimensä veroisia, joihin voi kirjoittaa kaikkea hyödyllistä tietoa reseptistä. Koonti vaatii, että kaikilla resepteillä on jokin lisenssi, joten sellainen tarvitsee laittaa reseptille. Tämä siksi, että Yocto-koonnista tuleva output-tiedosto sisältää tmp/deploy/licenses-kansion, josta näkee kaikkien ohjelmistojen lisenssit, jotka on sisälletty koontiin. SRC_URI-muuttujalla kerrotaan missä skriptit jotka halutaan asentaa on. Huomion arvoista, että vaikka skriptit ovat files nimisessä kansiossa niin tässä määritellään, että skripti löytyy file://-polusta. S-muuttuja, joka on nimetty vähän huonosti tarkoittaa koonnin polun sijaintia, eli missä pakkaamattomien reseptien lähdekoodit sijaitsevat. Workdir-sijainti tarkoittaa tässä tapauksessa build/tmp/work-kansiota, koska koonti tehdään build nimiseen kansioon. Do_install-funktio asentaa skriptit datadir_native-polkuun, joka vastaa /usr/share sijantia ja siihen lisäksi mender/inventory eli /usr/share/mender/inventory, minne laitetaan kaikki inventaariot. Install-työkalu siirtää tiedoston work-kansiosta /usr/share/mender/inventory-polkuun ja sille annetaan suoritus oikeudet.

Koonnin tekeminen tehdään siten, että ensin tarvitsee alustaa koontijärjestelmä. Tähän on olemassa valmis skripti nimeltä oe-init-build-env. Navigoidaan Yocto-kansioon, jossa ajetaan alustus-skripti seuraavasti:

```
- $ source src/poky/oe-init-build-env build
```

Tällöin voidaan määrittää, että koonti tehdään build-kansioon. Käyttämällä yhtä kansiota, minne koonti tulee, helpottaa asioita, esimerkiksi jos tarvitsee koota Yocto usealle eri raudalle, tällöin riittää, että jokaiselle alustalle tehdään oma koontikansio.

Yocto-projektissa build/conf/bblayers.conf-tiedostoon lisätään ne kerrokset, jotka halutaan lisätä käännökseen. Lisäyksen voi tehdä ajamalla bitbake-layers add-layer -komento ja lisäksi polku lisättävään kerrokseen.

Esim. bitbake-layers add-layer home/$USER/Yocto/src/meta-mender/meta-mender-raspberry.

```
random@Yocto:~/Yocto/build/conf$ cat bblayers.conf
# POKY_BBLAYERS_CONF_VERSION is increased each time build/conf/bblayers.conf
# changes incompatibly
POKY_BBLAYERS_CONF_VERSION = "2"

BBPATH = "${TOPDIR}"
BBFILES ?= ""

BBLAYERS ?= " \
  /home/random/Yocto/src/poky/meta \
  /home/random/Yocto/src/poky/meta-poky \
  /home/random/Yocto/src/poky/meta-yocto-bsp \
  /home/random/Yocto/src/meta-raspberrypi \
  /home/random/Yocto/src/meta-mender/meta-mender-core \
  /home/random/Yocto/src/meta-mender/meta-mender-raspberrypi \
  /home/random/Yocto/src/meta-mender/meta-mender-demo \
  /home/random/Yocto/src/meta-watchdogservice \
  "
```

Kuva 10. Työn bblayers.conf-tiedoston sisältö.

Tämän jälkeen tarvitsee vielä lisätä build/conf/local.conf-tiedostoon tarvittavat konfiguraatiot.

Local.conf-tiedostoon määritellään kaikki omat lisäykset, reseptit ja muut konfiguraatiot, joita tarvitaan esim. tässä työssä tarvittiin Mender-palvelimen asetukset.

```
#Määritetaan alusta joka on käytössä

MACHINE = "raspberrypi3"

#Julkaisu versionumero jota on tarkoitus aina kasvattaa, kun tulee uusi käännös

MENDER_ARTIFACT_NAME = "release-1"

#Konfiguraatio joka lisää yleisimmät Mender ominaisuudet

INHERIT += "mender-full"

#Otetaan uboot käyttöön, tämä asetus oli meta-mender-raspberrypi käännös ohjeissa

RPI_USE_U_BOOT = "1"

#Osiointi, nämä asetukset olivat meta-mender-raspberrypi käännös ohjeissa

MENDER_BOOT_PART_SIZE_MB = "40"

MENDER_PARTITION_ALIGNMENT = "4194304"

#Jotta voidaan päivittää Linux kernel, Mender käyttää levykuvaa joka sijaitsee root-
#tiedostojärjestelmässä ja varmistetaan, että asennetaan /boot kansioon.

#Tämä asetus oli meta-mender-raspberrypi käännös ohjeissa

IMAGE_INSTALL_append = " kernel-image kernel-devicetree"

#Ei käytetä rpi-sdimg vaan Mender:n levykuvaketta sdimg

#Tämä asetus oli meta-mender-raspberrypi käännös ohjeissa

IMAGE_FSTYPES_remove += " rpi-sdimg"

#Root-tiedostojärjestelmän tiedostomuoto

SDIMG_ROOTFS_TYPE = "ext4"

#Mender palvelimen IP-osoite

MENDER_DEMO_HOST_IP_ADDRESS = "192.168.1.199"

#Otetaan systemd käyttöön ja varmistetaan ettei sysvinit ole käytössä

#Mender palvelimen vaatimus, että  systemd pitää olla käytössä

DISTRO_FEATURES_append = " systemd"
```

```
VIRTUAL-RUNTIME_init_manager = "systemd"

DISTRO_FEATURES_BACKFILL_CONSIDERED = "sysvinit"

#Lisätään omat reseptit

IMAGE_INSTALL_append = " watchdogdeamon"

IMAGE_INSTALL_append = " mywatchdogservice"

IMAGE_INSTALL_append = " menderinventory"
```

Koonti suoritetaan bitbake-komennolla, ja argumentiksi asetetaan k, jotta mahdollinen virhe ei lopeta koontia, esim. tilanteessa jossa koonti jätetään päälle ja itse tehdään jotain toista asiaa sillä aikaa. Tämä on hyödyllinen ominaisuus, koska jos laittaa koonnin päälle ja menee itse lounaalle, sillä välin, niin virheen sattuessa, koontia ei lopeteta, vaan hypätään virheen aiheutuneen asian yli ja jatketaan prosessia.

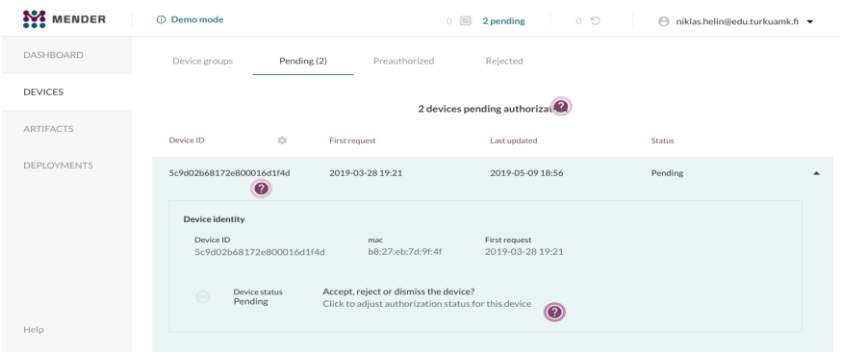
```
- $ bitbake -k core-image-full-cmdline
```

4.2.4 Testaaminen

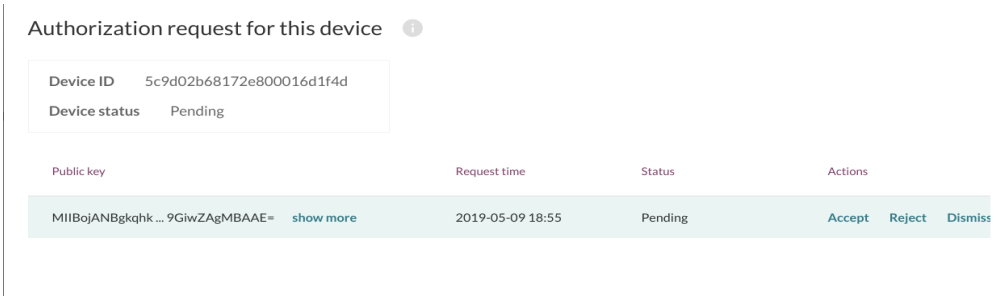
Levykuva löytyy /build/tmp/deploy/images/raspberrypi3-polusta ja tiedosto joka halutaan polttaa muistikortille, on core-image-full-cmdline-raspberrypi3--2019*.sdimg.

Levykuva voidaan polttaa muistikortille monella eri työkalulla esim. dd tai Etcher. Tämän operaation jälkeen asetetaan muistikortti Raspberry Pi:hin ja kytketään Ethernet kaapeli ja virtakaapeli kohdelaitteeseen.

Käynnistetään Mender-palvelin, kirjaudutaan käyttöliittymään, joka on osoitteessa https://localhost.Mender, palvelimella on nyt kaksi laitetta, jotka pyytävät hyväksyntää (Kuva 11), toinen on Raspberry Pi ja toinen on testiversion generoima testilaite.

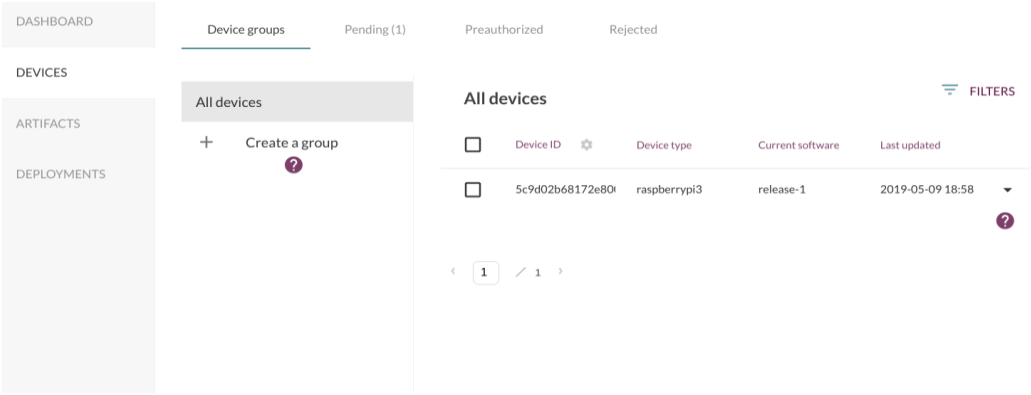


Kuva 11. Menderin käyttöliittymässä, kaksi laitetta odottavat hyväksyntää.



Kuva 12. Lisätään laite palvelimeen hyväksymällä se.

Laitteen hyväksynnän jälkeen laite on nähtävissä devices-osiossa.



Kuva 12. Hyväksytyt laitteet.

Testataan, että inventaariossa on failed_ssh ja uptime painamalla last updated -kohdan alla olevaa alaspäin osoittavaa nuolta, jolloin näkymä näyttää kuvan 13 mukaiselta.

Kuva 13. Kohdelaitteen inventaario.

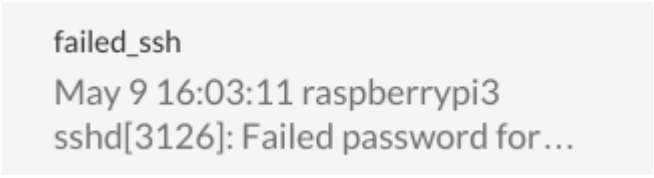
Todennettua, että failed_ssh- ja uptime-atribuutit löytyvät, testataan kirjautua SSH:n avulla kohdelaitteelle siten, että syötetään väärä salasana, jotta saadaan testattua, toimiiko inventory-skripti.

Laitteen IP-osoitteen voidaan selvittää käyttämällä arp-scan nimistä työkalua, kuvan 14 mukaisesti.

```
random@Yocto:~$ sudo arp-scan -l
[sudo] password for random:
Interface: enp3s0, datalink type: EN10MB (Ethernet)
Starting arp-scan 1.9 with 256 hosts (http://www.nta-monitor.com/tools/arp-scan/
)
192.168.1.1     50:c7:bf:c5:fa:67       (Unknown)
192.168.1.118   b8:27:eb:7d:9f:4f       Raspberry Pi Foundation
```

Kuva 14. Raspberry Pi:n IP-osoite.

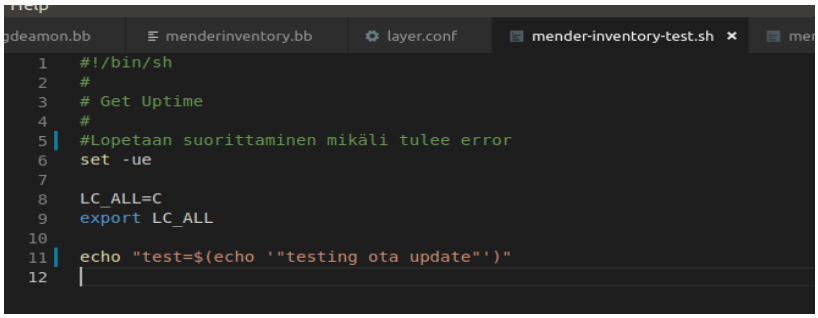
Terminaalissa muodostetaan SSH-yhteys, ajamalla komento ssh root@192.168.1.118 ja ohjelman pyytäessä salasanaa, kirjoitetaan väärä salasana. Huomataan muutaman sekunnin jälkeen, että failed_ssh attribuutti on päivittynyt joka osoittaa, että skripti toimii halutulla tavalla.



Kuva 15. Osoitus, että failed_ssh-atribuutti toimii.

Luonnollisesti kun kysymyksessä on Mender, joka on etäpäivityksiä mahdollistava ohjelmisto niin on hyvä testata päivityksen tekemistä Mender-palvelimen avulla. Päivitys on tässä työssä vain uuden Menderin inventory-attribuutin lisääminen. Attribuutin speksit laitettaan erittäin vaatimattomaksi ja tarkoitus on vain testata päivitystä.

Tehdään uusi skripti, kuvan 16 mukaisesti, nimeltä mender-inventory-test.sh, jossa printataan tekstiä Menderin inventory-attribuuttiin. Tämä lisätään menderinventory.bb- reseptiin, josta löytyy myös failed_ssh- ja uptime-skriptit.



Kuva 16. Uusi skripti.

Lisäksi muokataan local.conf-tiedostoa siten, että muutetaan:

- Vanha: MENDER_ARTIFACT_NAME=”release-1”
- Uusi: MENDER_ARTIFACT_NAME=”release-2”

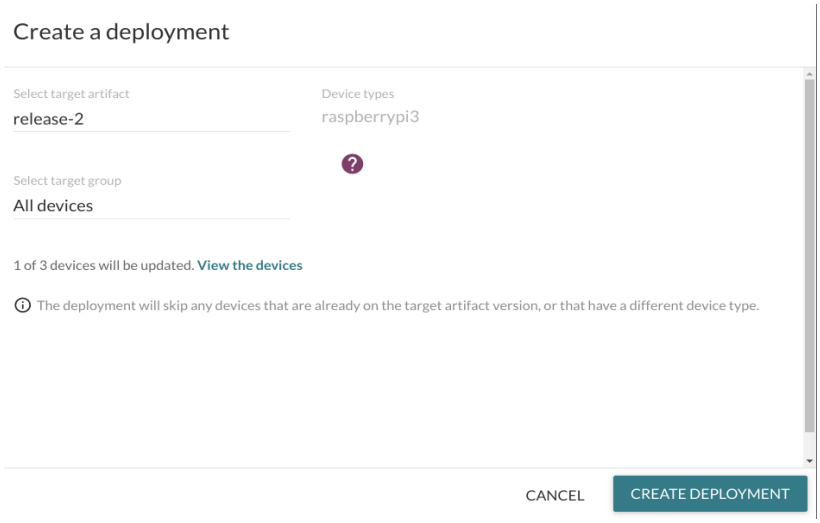
Tämän jälkeen ajetaan terminaalissa bitbake -k core-image-full-cmdline, jolla tehdään uusi koonti. Tällä kertaa tarkoitus ei ole enää kirjoittaa uutta levykuvaa muistikorttiin, vaan käytetään hyväksi Mender-kerroksen tuottamaa *.mender-tiedostoa, joka sisältää artefaktin. Todellisuudessa työssä tuli tehtyä toistakymmentä päivitystä, jonka takia käytössä oli skripti, joka kopio .mender-tiedoston scp-työkalun avulla Mender-palvelimelle.

Skripti joka kopio artefaktin palvelimelle:

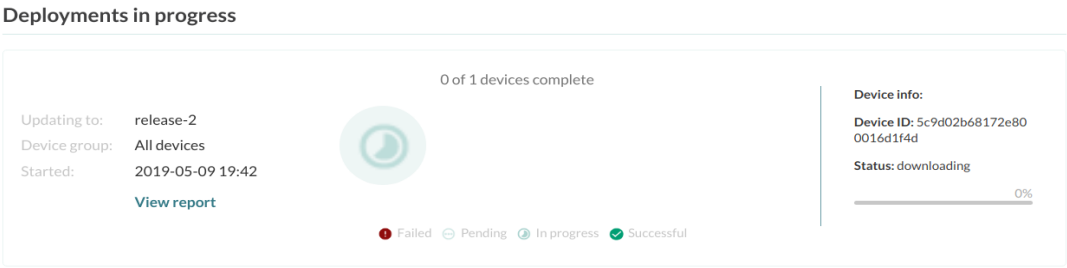
```
#!/bin/bash

cp      /home/random/Yocto/build/tmp/deploy/images/raspberrypi3/core-image-full-cmdline-raspberrypi3--
*.mender /home/random/

scp core-image-full-cmdline-raspberrypi3--*.mender mender@192.168.1.199:/home/mender/patch.mender
```

Kuva 17. Skripti kopio .mender-tiedoston palvelimelle

Päivitystiedosto ladataan Mender-palvelimille siten, että Artifacts sivustolle raahataan tai selaamalla tiedosto ladataan .mender-tiedosto palvelimelle. Tämän jälkeen valitaan Deployments-sivusto ja painetaan Create Deployment -nappulaa (Kuva 18) ja täytetään avautuvasta valikosta tiedot, joita pyydetään. Valitsemalla artifact rajataan heti, että laitteen tyyppi on Raspberry Pi 3 ja ryhmiä ei ole, joten valitaan kaikki laitteet. Päivitystä voi seurata Dashboard-sivustolta (Kuva 19), jossa näkee päivityksen etenemisen.



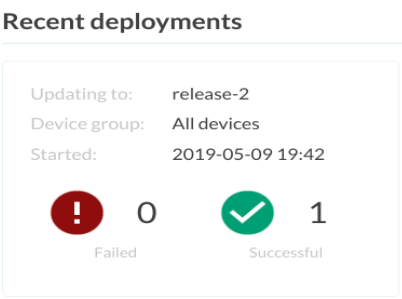
Kuva 18. Uuden päivityksen tekeminen Menderissä.



Kuva 19. Dashboard-sivustolta voidaan seurata päivityksen kulkua.

Päivityksen tekemisessä oli aluksi vaikeuksia ja kaikki päivitykset päättyivät epäonnistumiseen. Ongelma johtui siitä, että laitteessa oli kellon kanssa ongelmia, tämä korjattiin siten, että ajettiin bitbake tzdata -komento ja lisättiin local.conf-tiedostoon: IMAGE_INSTALL += "tzdata-europa", jolla lisätään aikavyöhyke juuritiedostojärjestelmään. Tämän jälkeen päivitykset onnistuivat.

Päivityksen oltaessa valmis ilmestyy kojelautaan tieto siitä, että onnistuiko päivitys.

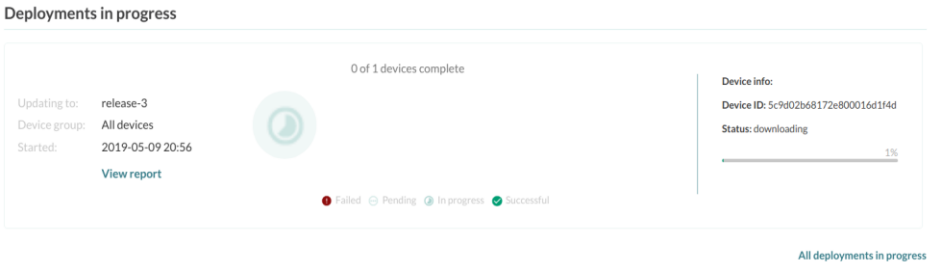


Kuva 20. Onnistunut päivitys.

Katsottaessa laitteen inventaariota huomataan, että artifact_name on nyt release-2 ja lisäksi test-atribuutti on lisättynä.

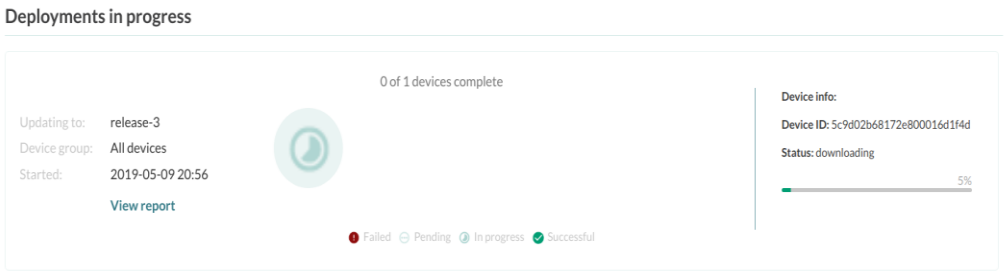
Kuva 21. Päivittynyt inventaario, jossa on test-attribuutti.

Lopuksi työssä testataan päivittää laite, mutta ennen päivityksen valmistumista vedetään virtajohto hetkeksi pois ja laitetaan sitten takaisin. Testin tarkoitus on kokeilla Menderin päivityksen luotettavuutta. Haluttu lopputulos olisi, että minkäänlaista vahinkoa ei tapahdu, vaan laite käynnistyisi toimista levyosiosta.



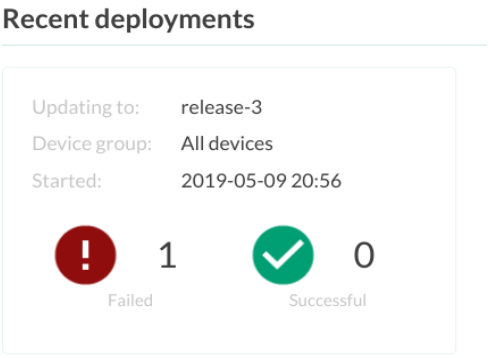
Kuva 22. 1 %:n kohdalla otetaan virta pois.

Päivityksen progressipalkki menee eteenpäin, vaikka laite ei ole edes päällä. Viiden prosentin kohdalla laitetaan virta päälle.



Kuva 23. 5 %:n kohdalla laitetaan virta takaisin päälle.

Päivitys jatkaa menoa eteenpäin ja loppuu samalla tavalla kuin onnistunut päivitys, mutta kun katsotaan mitä oikeasti tapahtui niin odotetusti päivitys on epäonnistunut.



Kuva 24. Epäonnistunut päivitys.

Päivityksestä löytyy lokitiedosto, mutta sitä selaamalla ei pysty todentamaan muuta kuin sen, että yhteys lakkasi toimimasta.

5 YHTEENVETO

Tavoitteena oli tehdä Yoctolla Raspberry Pi 3:een Linux-levykuva ja mahdollistaa Menderillä järjestelmän etäpäivitys. Tavoitteessa onnistuttiin ja testaamalla varmistettiin, että lopputulos oli halutunlainen. Lisäarvoa olisi saatu, mikäli yhteys Mender-palvelimen ja Raspberry Pin välillä oltaisiin saatu toimimaan tuotantoversiolla testiversion sijaan.

Jatkokehityksenä olisi ollut, että olisi käytetty Qt-kerrosta tai WWW-palvelinta hallintapaneelin tekemiseen kohdelaitteeseen, koska oikeassa skenaariossa päivitykset halutaan usein suorittaa vasta hyväksynnän jälkeen, eikä siten, että päivitys tapahtuu ilman, että siitä tulee mitään ilmoitusta. Laite voi olla tekemässä jotain kriittistä asiaa, esim. auton ohjelmistoa ei voi päivittää ajon aikana.

Aihe oli henkilökohtaisesti mieluinen, sillä Linux on itselleni tarjonnut monia työpöytäympäristössä ikimuistoisia onnistumisen hetkiä kuin myös masentavia päiviä. Nyt pääsin käyttämään oppimaani ja lisäksi kerryttää uutta tietoa, miten sulautettuun laitteeseen voidaan luoda Yocto-projektilla spesifisti Linux-käyttöjärjestelmä. Lisäksi pääsin tutustumaan ja opettelemaan Menderin käyttöä.

LÄHTEET

[1] Will the new EU standard protect consumers from IoT products? [Viitattu 20.5.2019] https://internetofthingsagenda.techtarget.com/blog/IoT-Agenda/Will-the-new-EU-standard-protect-consumers-from-IoT-products

[2] What is Mender? [Viitattu 1.4.2019] https://mender.io/overview/what-is-mender

[3] Hidden Costs of Homegrown Updaters [Viitattu 1.4.2019] https://mender.io/learn/whitepapers/_resources/hidden-costs-of-homegrown

[4] Mender document 1.7 [Viitattu 7.4.2019] https://docs.mender.io/1.7/architecture/overview

[5] Mender document 1.7 [Viitattu 7.4.2019] https://docs.mender.io/1.7/architecture/mender-artifacts

[6] Getting Started: The Yocto Project Overview [Viitattu 6.4.2019] https://www.yoctoproject.org/software-overview/

[7] Simmonds, C. Mastering. Embedded Linux Programming. United Kingdom: Packt Publishing, 2016. 418 s. ISBN 978-1-78439-253-6

[8] Rifenbark,S Yocto Project Mega-Manual [Viitattu 7.4.2019] https://www.yoctoproject.org/docs/2.6/mega-manual/mega-manual.html

[9] BitBake [Viitattu 6.4.2019] https://en.wikipedia.org/wiki/BitBake

[10] Erway,T & Moseley,D Is Yocto Project for you [Viitattu 7.4.2019] https://www.yoctoproject.org/is-yocto-project-for-you/