Valeri Bart

Rekisterikilpitunnistus DOCKER-ympäristössä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

04.04.2017

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Valeri Bart Rekisterikilpitunnistus DOCKER-ympäristössä 30 sivua 04.04.2018
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikan koulutusohjelma
Suuntautumisvaihtoehto	Tietoverkot
Ohjaaja(t)	Lehtori Harri Ahola Osaamisaluepäällikkö Janne Salonen

Insinöörityössä laajennettiin rekisterikilpien tunnistukseen tarkoitettua OpenALPR-kirjastoa ohjelmoimalla siihen tietokantatoiminto. Asennettiin MySQL-tietokantapalvelin ja saatiin OpenALPR-agentti toimimaan tietokannan kanssa. Sen jälkeen selvitettiin, kuinka aikaansaatu kokonaisuus voidaan virtualisoida Docker-tekniikalla. Käytettiin kahta PC:tä, joista toiseen oli asennettu Ubuntu- ja toiseen (jossa tapahtui virtuaalisointi) Fedora- käyttöjärjestelmä.

Avainsanat	Docker, virtuaaliympäristö, OpenALPR, MySQL

Author(s) Title Number of Pages Date	Valeri Bart License Plate Recognition in Docker-environment 30 pages 04 April 2018
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Data Networks
Instructor(s)	Harri Ahola, Senior Lecturer Janne Salonen, Head of Department

In the thesis, the OpenALPR, library for the identification of license plates, was extended with the database functionality. A MySQL database server was installed and it were made OpenALPR agent work with the database. It was then inspected and studied how the achieved entity can be virtualized using Docker technology. Two PCs were used, one with Ubuntu and one (for Virtualization) with Fedora.

Keywords	Docker, virtualized environment, OpenALPR, MySQL

Sisällys

Liitteet

Liite 1. Insinöörityössä käytetty laitteisto

Liite 2. Tiedostoon daemon.cpp tehdyt muutokset

Lyhenteet

ALPR	Automatic License Plate Recognition – automaattinen rekisterikilpien tunnistus.
CV	Computer Vision - tietokonenäkö

1 Johdanto

Tietokonenäkö on tietotekniikan osa-alue, jota on käytetty jo useita vuosikymmeniä ajoneuvojen rekisterikilpien tunnistamiseen ympäri maailmaa. Ja kuten monia muita tietotekniikkaa hyödyntäviä teknologioita, automaattista rekisterikilpien tunnistusta on mahdollista virtualisoida ja virtualisoinnin ansiosta optimoida tunnistukseen käytettäviä resursseja tai käyttää pilvipalveluja resursseina. Työkaluksi työn tunnistusosuuden toteuttamiseen on valittu OpenALPR-kirjasto, joka pohjautuu sovelluskehittäjien maailmassa tunnettuun OpenCV-kirjastoon. Virtuaalisointiteknologiaksi valittiin Docker, mm. sen paremman toimintanopeuden takia. Työssä tutustuttiin em. teknikoihin ja saatiin ne toimimaan yhdistettyinä toisiinsa. Kuvan analysointi vaatii paljon prosessoritehoa, mikä asettaa tiettyjä vaatimuksia laitteistolle, mutta vaihtoehtona yhden kalliin tietokoneen ostamiselle voi olla useamman vanhemman tietokoneen käyttäminen, minkä virtuaalisointi ja Docker tekee mahdolliseksi.

2 Lainsäädäntö

IP-kamera on digitaalinen videokamera, laite, jolla voidaan kuvat mm. ihmisiä. Mikäli kameralla saatuja kuvia pyritään tallentamaan, tulee henkilötietolakia huomioida. Kun videovalvontaa toteutetaan ympäristössä, missä kuvaan voi ilmestyä ihmisiä siten, että heidän kasvot ovat kuvasta tunnistettavissa ja kuvia tallennetaan, niistä muodostuu henkilötietorekisteri. Tietokantaan tallennettavista tiedoista, kuten kuvan tallennusajankohta ja kameran sijainti, myös muodostuu henkilötietorekisteri, jos samalla otetaan kuvia talteen.

Jos henkilötietorekisterin muodostaminen on mahdollista, videovalvontajärjestelmän käyttäjällä on velvollisuus asentaa valvonnasta varoittava kyltti valvonta-alueen hyvin näkyvään paikkaan. Varoituskyltin tehtävä on ilmoittaa kameroiden ohi kulkeville ihmisille siitä, että kyseisessä paikassa tapahtuu videovalvonta. Mikäli jokin yritys käyttää videovalvontajärjestelmää esimerkiksi omalla parkkipaikalla, tulee voimaan laki yhteistoiminnasta, jonka mukaan työnantaja on velvollinen ilmoittaa omille työntekijöille heihin kohdistuvasta valvonnasta ja sen luonteesta yhteistoimintamenettelyssä. [ST-kortti 664.10]

Rikoslain salakatselusäännöstä ei voi soveltaa tässä työssä käsiteltävän järjestelmän käyttäjään, mikäli valvonta-alueella on yllä mainittu kyltti hyvin näkyvässä paikassa. Tässä työssä käsiteltäviltä kameroilta ei kuitenkaan ollut tarkoitus saada kuvia nähtäväksi tai tallennettavaksi.

3 Virtuaalisointi

Virtuaalisonnista saatavia hyötyjä ovat mm. laitteistoresurssien käytön optimointi, prosessien turvallinen ajo ja teknisen tuen tarpeen väheneminen. Tästä kaikesta seuraa myös kustannusten vähenemisen mahdollisuus. Ilman virtuaalisointia, tyypilliset sovellukset (esim. yhdistelmä Apache + PHP + MySQL) käyttävät vain osaa prosessorin tehosta, mutta virtuaalisoinnin avulla samat prosessit voidaan skaalata, jolloin prosessorin tehosta saadaan käytettyä sata prosenttia. Resurssien optimointi yleisesti voi olla kaksisuuntainen. Toisaalta voidaan optimoida yksittäisen tietokoneen prosessoritehon käyttöä käynnistämällä samalla tietokoneella useita virtuaalisia käyttöjärjestelmiä. Toinen optimointisuunta on useamman kuin yhden tietokoneen resurssien käyttö yhden prosessin suorittamiseen, mikä ei ehkä liity suoraan virtuaalisointiin.
Käyttämällä virtuaalisoitua ympäristöä saadaan prosessi, joka on tämän työn tapauksessa videokuvan analyysi, mahdollisimman riippumattomaksi tietokoneen käyttöjärjestelmän tyypistä, versiosta ja siihen asennetuista kirjastoista. Virtuaalisoinnin ansiosta analyysiprosessia voidaan toteuttaa käyttäen samanaikaisesti useiden käyttöjärjestelmien etuja, useita laitteistotyyppejä ja laiteresursseina voivat olla sekä lähiverkossa olevat laitteet, että pilvestä saatavat palvelut. Resurssien laajennus on virtuaalisoidussa ympäristössä kustannustehokkaampaa, miksei helpompaakin, kuin perinteisessä.

4 Docker

4.1 Yleistä

Kontit (Containers)ovat yleisesti ottaen tiedostojärjestelmän näennäisiä kopioita, joita kytketään käynnistettävien sovellusten tiedostojärjestelmiksi. Hyötynä sellaisesta ajotavasta on se, että kontissa ajettava sovellus, tarkoituksella tai vikaantuessa, ei pysty vahingoittamaan varsinaista isäntätietokoneen tiedostojärjestelmää. Kontti on muuten suhteellisen vanha eristyksen konsepti. Chroot-komento, joka luo eristetyn tiedostojär-

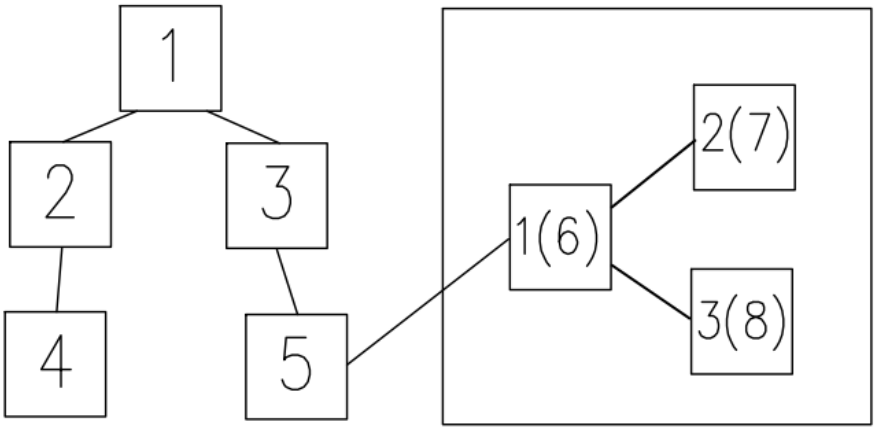
jestelmälohkon sovellukselle, on ollut UNIX-pohjaisissa käyttöjärjestelmissä jo vuodesta 1979. Jail, joka on chroot-konseptin laajennus FreeBSD:ssä, on ollut olemassa vuodesta 1998. Solaris Containers (nykyisin Solaris Zones), Solaris-käyttöjärjestelmän konttien vastine, esitettiin vuonna 2001. [Using Docker]

Hewlett Packard esitti tekniikkansa nimeltä Secure Resource Partitions for HP-UX (HP-UX Containers) vuonna 2007 ja vuonna 2008 Linux Containers (LXC) esitettiin maailmalle.

Docker on suhteellisen uutta tekniikkaa, jonka Solomon Hykes, Dockerin tekijä, esitti Python-kehittäjien konferenssissa 15 maaliskuuta 2013, Santa Clarassa, Californiassa. Dockeria voidaan sanoa käyttäjäystävälliseksi Linuxin ytimen ohjausmekanismien käyttöliittymäksi. Seuraavaksi kuvataan lyhyesti kyseessä olevat dockerin hyödyntämät ytimen mekanismit:

cgroups – prosessoriajan, muistin ja tallennustilan annostusmekanismi

namespaces – nimiavaruudet, mekanismi, jonka avulla eristetyt prosessit ”luulevat” hallitsevansa yksilöllisesti verkkoliittymiä, käyttäjätietokantaa, muiden prosessien kanssa kommunikointia jne. Sillä tekniikalla voidaan luoda eristettyjä prosessipuita, joissa on omat prosessinumeroa yksi omaavat init-prosessit.

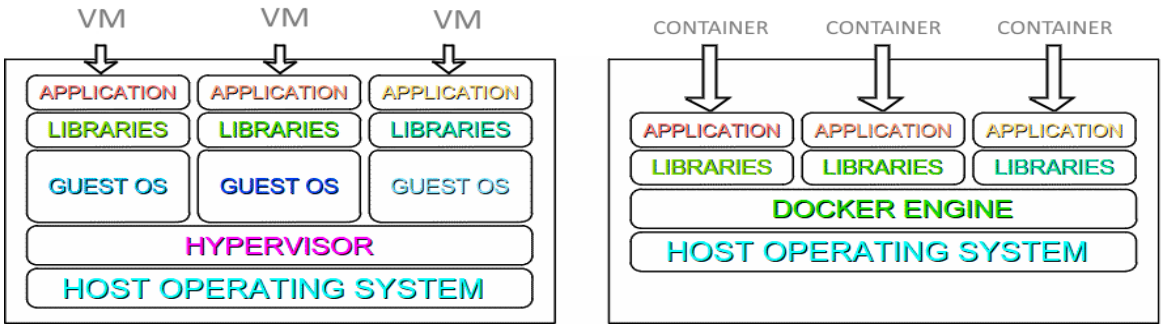


Kuva 1. Process namesapaces. Numerot ovat prosessitunnuksia.

Network namespaces – verkkorajapinnan eristys, jonka ansiosta jokaisella prosessilla on oma porttiavaruus ja reititystaulu.

SE Linux – laajennettu ja tarkempi versio Linuxin perinteisestä käyttäjä- ja järjestelmäoikeuksien hallintatekniikasta.

Laskentanopeus on kriittinen tekijä videoanalyysiprojekteissa, joten virtuaalisointiteknologiaksi tähän työhön valittiin Docker, koska Docker-kontit ovat nopeampia ja resurssikevyempiä virtuaalikoneisiin verrattuna. Docker-konttien ja virtuaalikoneiden (esim. VMware) tärkein periaatteellinen ero selvinnee seuraavasta kuvasta:



Kuva 2. VMwaren ja Dockerin ero.

Kuten kuvasta voidaan nähdä, Dockerin tapauksessa vieraskäyttöjärjestelmän kerros puuttuu kokonaan. Sen lisäksi kontit ovat resurssikevyempiä, koska ne eivät ole kokonaisten käyttöjärjestelmien kopioita, kuten virtuaalikoneet. Nämä ovat suurimmat syyt konttien nopeampaan toimintaan virtuaalikoneisiin verrattuna. Docker toiminnallisuudeltaan ei vastaa myöskään hypervisoria, vaan se on helppokäyttöinen ja turvallinen järjestelmän ydinpalvelujen jakomekanismi.

4.2 Levykuva

Docker-image (image) – on levykuva, joka toimii mallina konttien luomiseen. Se rakennetaan docker build –komennolla, luomalla ensin Dockerfile-tiedosto, jonka jokainen käskyrivi luo oman lisäkerroksen pohjakuvaan. Pohjakuvaksi sanotaan yleensä minimalistista levykuvaa jostakin käyttöjärjestelmästä, johon lisätään vain tietyn tehtävän suorittamiseen tarvittavia paketteja ja tiedostoja. Kun image on kerran rakennettu

docker build –komennolla, se pysyy muuttumattomana, kunnes se poistetaan. [Up and running].

4.3 Kontti

Kontti (Docker-container) – ajettava kopio levykuvasta. Kuten jokaisessa ajettavassa prosessissa ja toisin kuin levykuvien sisällä, kontin sisäisessä avaruudessa voi tapahtua tiedostojärjestelmämuutoksia. Kontin suurin ero levykuvaan nähden on se, että kontissa on kirjoitettava kerros, jota sanotaan konttikerrokseksi ja joka hävitetään kontin lopetuksen yhteydessä.

4.4 Rekisteri

Rekisteri on verkon kautta käytettävissä oleva julkinen tai yksityinen levykuvien kokoelma, mistä levykuvia voidaan ladata vapaasti tai kirjautumisen jälkeen. Esimerkkeinä julkisista rekistereistä voivat olla Docker.io ja Quay.io.

4.5 Tietoturva

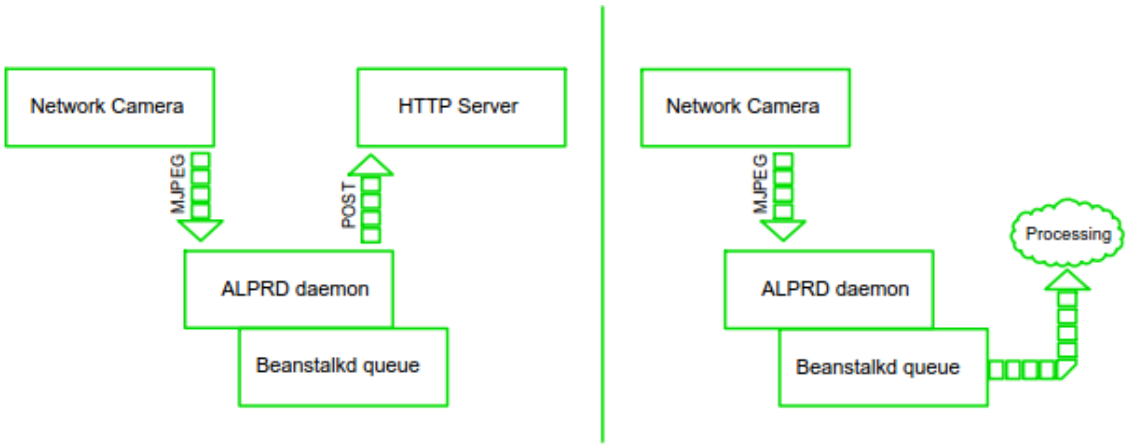
Ihmisen ohjelmoimasta koodista on aina ilmennyt suurempia tai pienempiä tietoturva-aukkoja, joita ei ole huomattu koodin ohjelmointi- tai testausvaiheessa. Tämä koskee myös Docker-konteissa käynnistettäviä sovelluksia. Konttien eristystä ei pidetä niin korkeantasoisena kuin virtuaalikoneiden eristystä, minkä takia dockerin tietoturvariskiä pidetään korkeampana. Docker-palvelu jakaa konteille isäntäkoneen ydinfunktioita eristettyinä ja syynä heikompaan dockerin eristystasoon on mm. se, että kaikki ytimen toiminnot eivät ole nykypäivänä eristettävissä. Yksi Dockerin tietoturvaongelmista on mm. se, että docker-palvelua ajetaan aina root-oikeuksilla, koska muuten eivät ytimen eristysfunktiot ole hyödynnettävissä. Jos ytimestä löytyy esimerkiksi viallinen funktio ja tietomurtaja saa jonkin kontin haltuunsa, hän pääsee niin sanotusti eristyksen yli ja pystyy saamaan haltuunsa koko isäntäkoneen. [Up and Running].

5 OpenALPR

OpenALPR (Open Automatic License Plate Recognition) on avoimen lähdekoodin, autojen rekisterikilpien tunnistukseen suunniteltu kirjasto, joka on julkaistu lisenssillä GNU AfferoGPLv3. Se on ohjelmoitu C++-ohjelmointikielellä, mutta siinä on rajapinnat C#-, GO-, Java- ja Python-kieliin. Kirjastoon kuuluu mm. OpenALPR-agentti - ohjelma, jota ajetaan taustalla ja joka suorittaa varsinaista kilpitunnistusta videovirroista.

OpenALPR-agentin rekisterikilpien tunnistustehokkuuden nyrkkisääntönä voidaan pitää, että pöytätietokoneille tyypillinen Intelin Core i5 - prosessori pystyy käsittelemään kahtakymmentä kehystä 720p-laatuisesta kuvavirrasta yhdessä sekunnissa. Toisin sanoen yksi agentti pystyy silloin palvelemaan kolmea kameraa, jos autot liikkuvat pienellä nopeudella, kahta kameraa jos autot liikkuvat keskinopeudella ja vain yhtä kameraa autojen nopeuden ollessaan korkea. Agentti käyttää fyysisellä tietokoneella 500 megatavua käyttömuistia per säie. [Nyrkkisääntö]

Säikeitä käynnistyy aina yksi per videovirta. Tämän työn luonteeseen kuuluu, että jokaiselle kameralle käynnistetään oma kontti, joka palvelee vain omaa kameraa, joten voidaan sanoa, että jokainen tunnistava kontti käyttää yli 500 megatavua käyttömuistia. Kontin käyttämän muistilohkon kokoa voidaan rajoittaa antamalla –memory-parametri halutulla muistikoolla. Jos rekisterikilpiä tunnistavana alustana on paikallinen tietokone, jossa on NVIDIA-näyttökortti, tunnistustehoa on mahdollista parantaa asentamalla OpenCL-kirjasto järjestelmään tai tämän työn tapauksessa integroimalla agentin levykuvioon. Em. kirjaston avulla OpenALPR pystyy hyödyntämään näyttökortin laskentaresursseja.

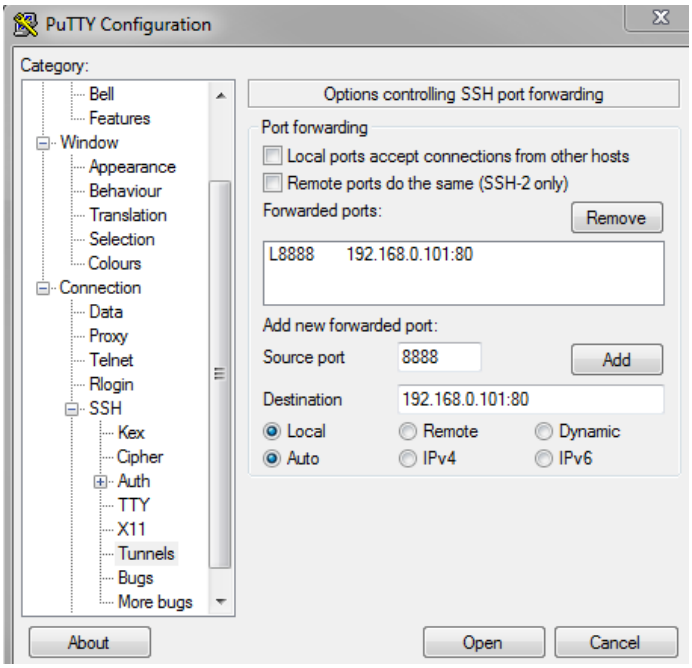


Kuva 3. Openalpr-tunnistuksen toimintavaihtoehdot.

6 Toteutus

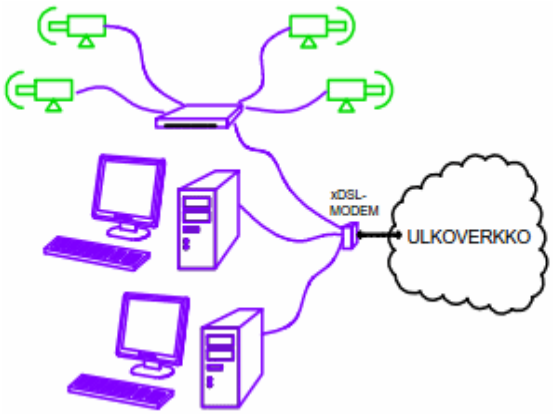
6.1 Laitteisto

Työn toteutusympäristönä käytettiin kahta PC-tietokonetta, joiden komponentit selviävät liitteessä olevista Linux-komentojen tulosteista. Annetut komennot olivat: cat /etc/os-release, uname –a, lspci, lscpu, lsblk, hwinfo –smartctl, cat /proc/meminfo. Kameroiden liitämiseen samaan verkkoon käytettiin Netgearin 5-porttista gigabittistä kotikäyttöön tarkoitettua peruskytkintä. Modeemina on toiminut ADSL2+/VDSL-modeemi TeleWell tw-eav510 v2. Kameroiden malli oli Opticam O3. Työtä oli tehty osittain etänä työpaikalta. Silloin käytettiin Putty-ohjelmaa, jolla nostettiin SSH-tunneli kotiin, jolloin puttylla avattiin yhteys loop-osoitteeseen 127.0.0.1 ja porttiin 8888, joka oli asetettu tunnelin lähtöportiksi. Modeemin asetussivulta selvitettiin modeemin julkinen IP-soite, joka laitettiin etäosoitteeksi. Modeemissa DMZ-palvelimeksi määritettiin aina toisen kotona olevan tietokoneen IP, sarjasta 192.168.0.XXX. SSH-palvelun portiksi kotikoneella määritettiin 80, koska työnantajan palomuuri ei sallinut yhteyksiä SSH:n oletusporttiin 22. Paikalliseksi tunnelin osoitteeksi valittiin 192.168.0.XXX:80 ja tunnelin tyypiksi local.



Kuva 4. Tunnelin asetukset

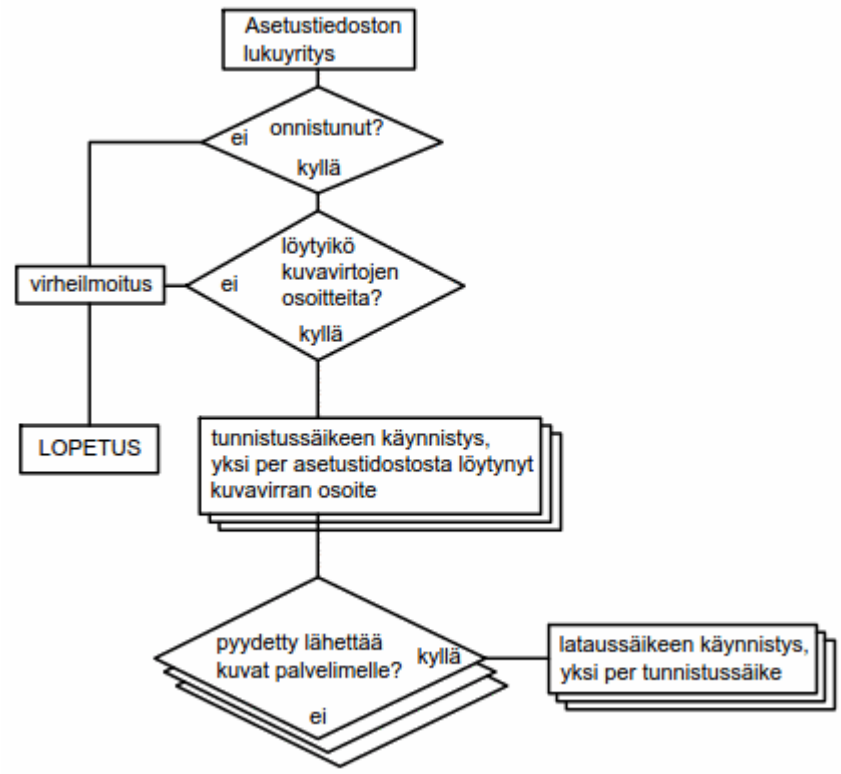
Kuva 5. Opticam O3



Kuva 6. Työssä käytetyn laitteiston kytkentäkaavio

6.2 OpenALPR:n täydennys

OpenALPR-kirjastoon kuuluu kaksi binääriohjelmaa, joista toinen on openalpr ja toinen alprd. Openalpr on sovellus, joka ottaa vastaan staattisen kuvan ja palauttaa siitä kuvasta löytämänsä rekisterinumerot. Alprd on taas taustalla ajettava linux-palvelu (daemon), joka etsii rekisterikilpiä kuvavirroista. Juuri sen lähdekoodia ruvettiin tutkimaan ja selvitettiin pääpiirteissä sen toimintalohkokaavio:



Kuva 7. Alprd-palvelun toimintakaava

Taustaprosessin edetessä käynnistyy tunnistussäikeitä, niin monta, kuin kuvavirtaosoitteita on asetustiedostossa. Tunnistussäikeissä tapahtuu varsinainen kilpitunnistus, jonka tuloksia sijoitetaan Beanstalk-jonoon ja lähetetään palvelimelle[1], mikäli sellainen on määritelty asetuksissa. Alprd voi myös tarvittaessa tallentaa kuvat, joista on tunnistettu kilpiä paikalliseen kansioon, mikäli sellainen on määritelty asetuksissa. Kaiken kaikkiaan openalpr-kirjaston lähdekoodista oli täydennetty 4 tiedostoa: daemon.cpp, daemonconfig.h, daemonconfig.cpp ja CMakeLists.txt

Tässä työssä puhutaan docker-konteista, joten mm. OpenALPR:stä tarvitaan erillinen levykuva. OpenALPR löytyykin docker-imagena kokoelmasta https://hub.docker.com. Sen sisällä on OpenALPR:n lähdekoodi, josta Dockerfile-tiedoston avulla kootaan varsinaiset suoritettavat ohjelmat alpr ja alprd. Koska alprd-daemoniin haluttiin lisätä tietokantatoiminto, piti muuttaa myös Dockerfile-tiedosto saadakseen tietokannan kehityskirjastot käyttöön:

[1] Maksulliseen OpenALPR-versioon, kuuluu mm. web-palvelu, joka käsittelee tunnistustuloksia ja lataa tarvittaessa kuvat löydetyistä kilvistä paikalliselta tietokoneelta.

```
FROM ubuntu:latest
# Install prerequisites
RUN apt-get update && apt-get install -y --no-install-recommends \
    build-essential cmake curl git libcurl3-dev \
    libleptonica-dev liblog4cplus-dev libopencv-dev libtesseract-dev \
    libmysqlcppconn-dev \
    libmysql++-dev \
    wget && \
    rm -rf /var/lib/apt/lists
# Copy all data
COPY . /srv/openalpr
# Setup the build directory
RUN mkdir /srv/openalpr/src/build
WORKDIR /srv/openalpr/src/build
# Setup the compile environment
RUN cmake -DCMAKE_INSTALL_PREFIX:PATH=/usr \
-DCMAKE_INSTALL_SYSCONFDIR:PATH=/etc .. && \
    make -j2 && \
    make install && \
    rm -rf /srv/openalpr
ENTRYPOINT ["alprd","-f"]
```

Viimeiseen riviin lisättiin –f-tarkenne, että voitaisiin nähdä tunnistuskonttien mahdolliset virhe- ja loki-ilmoitukset komennolla ”docker logs”.

Muutetun alprd-imagen koonti tapahtuu seuraavalla komennolla:

```
docker build –t alprd .
```

6.3 MySQL

Yksi perinteisistä tavoista tallentaa jonkin ohjelman laskentatuloksia on niitten sijoitus tietokantaan. Tällä tavalla edelleen toimivat ohjelmistomaailman tuhannet sovellukset. Mutta ilmainen OpenALPR-kirjaston osuus ei ole suunniteltu toimimaan tietokannan kanssa. Sen puutteen vuoksi OpenALPR:ään oli päätetty lisätä tietokantatoiminto, mikä

edellytti kirjaston alkuperäisen koodin muokkaamista. Ks. Liite 2 – tiedostoon daemon.cpp tehdyt muutokset.

Kun koodi, joka avaa yhteyden MySQL-tietokantaan ja lisää siihen tunnistuksessa löydettäviä rekisterinumeroita, oli alustavasti suunniteltu, se sijoitettiin tunnistussäikeen loppuosaan, missä tapahtui mm. tulosten sijoitus beanstalk-jonoon. Tietokanta-asetukset oli päätetty sijoittaa samaan tiedostoon alprd-agentin asetusten kanssa.

Asetukset, jotka oli lisätty:

db_addr = tietokannan url-osoite muodossa osoite:portti

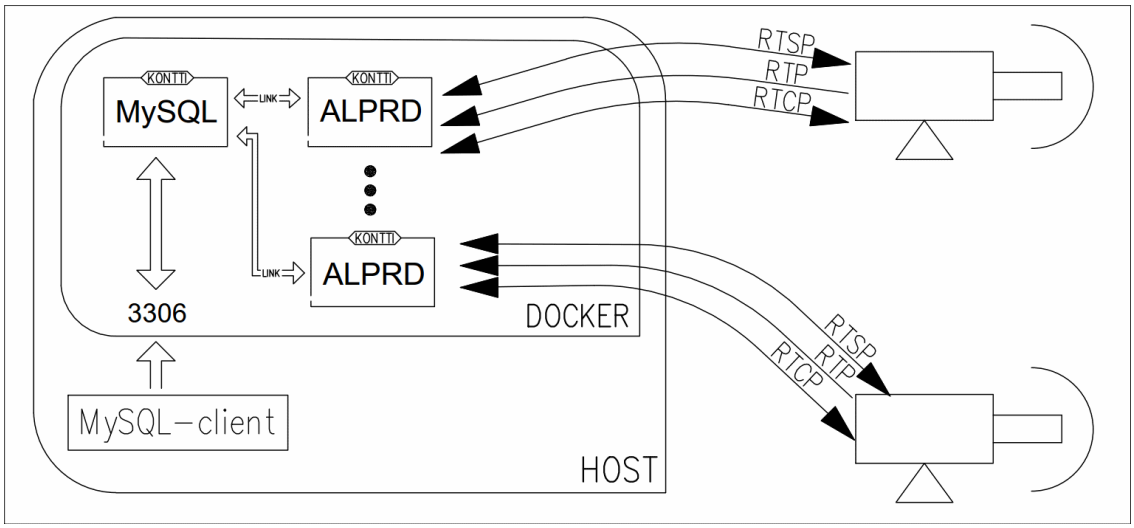
db_user = tietokantakäyttäjä

db_pwd = tietokantakäyttäjän salasana

db_cameraname = kameran tunniste

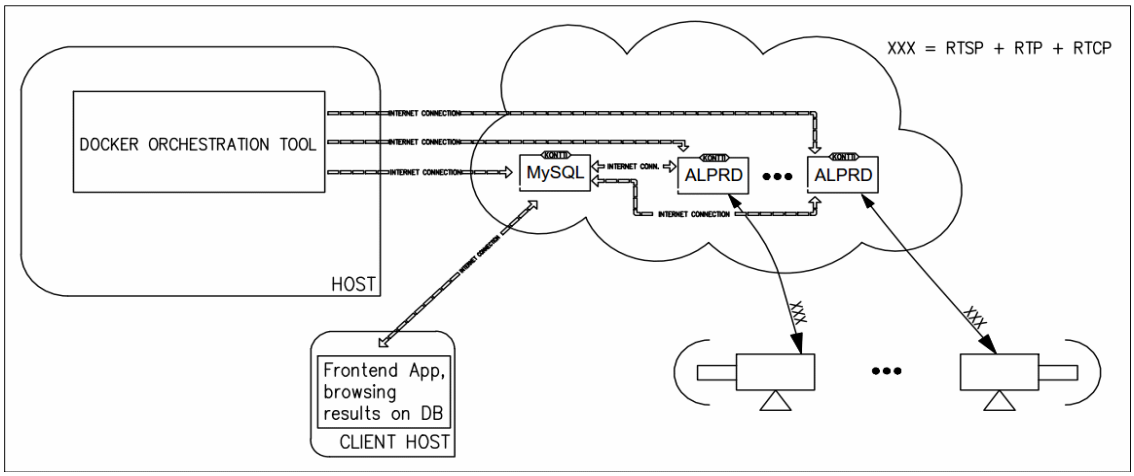
6.4 MySQL Connector

Docker-ympäristö ja siinä ympäristössä suoritettu ALPR-tunnistus on kokonaisuus, jossa MySQL-kontti on ainoa osa, jonka tulee näkyä ulkomaailmalle, että tapahtumia voitaisiin selata mistä fyysisestä paikasta tahansa, myös silloin kun koko tunnistusjärjestelmä toteutetaan yhdellä tietokoneella. Kuvassa 3 on esitetty täysin paikallisen kokoonpanon periaate.



Kuva 8. Tunnistus tapahtuu paikallisella tietokoneella

Muiden osien näkyvyyttä ulkomaailmalle ja fyysistä sijaintia taas voidaan säädellä aina resurssitilanteen ja tarpeiden mukaan. Toisaalta, paikallisella tietokoneella ei tarvitse välttämättä olla yhtään ajossa olevaa konttia vaan kaikkia kontteja voidaan ajaa pilvessä, riippuen käytetyn kaistan leveydestä (kuva 4). Siinä tapauksessa jokaista kameraa varten tulee avata aina kolme verkkoyhteyttä jokaiseen kameraan: RTSP-yhteys, jolla ohjataan median siirtoa, RTP-yhteys, jolla siirretään kuvaa ja RTCP-yhteys kuvan metatiedon (pakkaus- ja synkronointitietojen) siirtoa varten.



Kuva 9. Tunnistus tapahtuu pilvessä

Em. kolmea yhteyttä voitaisiin sulauttaa yhteen, esim. HLS- tai RTMP-protokollaa käyttäen, jolloin vaatimukset kaistanleveydelle ja verkkoturvasäännöille tulisivat yksinkertaisemmiksi. Mutta silloin kyseistä sulautusta toteuttavat sovellukset/kontit olisi silti

pakko ajaa kameroiden välittömässä läheisyydessä. Mikäli vaatimuksena on se, että kameroiden kanssa samassa lähiverkossa ei saa ajaa yhtään sovellusta tai konttia, voidaan hyödyntää pilvipalvelutarjoajan tarjottavaaVPN-toimintoa. Siinä tapauksessa kameraverkkoon pitäisi asentaa VPN-yhdyskäytävä, joka voisi olla esimerkiksi VPN-toiminnolla varustettu reititin. MySQL-tietokannan kanssa ohjelmalliseen operointiin tarvitaan ohjelmistorajapinta. Toisin sanoen, järjestelmään tai tässä tapauksessa OpenALPR:n levykuvaan, täytyy lisätä kirjasto, jossa on tarvittavat tietokantafunktiot. Tähän tarkoitukseen oli valittu mysql-cpp-connector. Valintaan vaikutti se, että mysql-cpp-connector on kirjoitettu C++-kielellä, samoin kuin OpenALPR, että sen voi asentaa "paketista" useimmissa Linux-distribuutioissa ja että sen käyttöohjeet löytyvät suoraan MySQL-projektin Web-sivulta. Että kirjasto voitiin koota lähdekoodista, muokattiin make-komennon ohjetiedostoa nimeltä CMakeLists.txt:

```
...

include_directories(./openalpr /usr/include)

...
IF (WITH_DAEMON)
...
                TARGET_LINK_LIBRARIES(alprd
                  ${OPENALPR_LIB}
                   support
                   video
                   curl
                   ${OpenCV_LIBS}
                   ${Tesseract_LIBRARIES}
                   ${log4cplus_LIBRARIES}
                   ${Extra_LIBS}
                   mysqlcppconn # <-- käännetyjen konnektorin funktioiden etsintäpolun osa
  )
ENDIF()
...
```

6.5 Tietokanta kontissa

Jokaisen kontin toiminnan olennainen ja ehkä merkittävin ominaisuus on se, että kontin lopetuksen jälkeen kaikki konttiin tehdyt muutokset häviävät. Toisin sanoen, jos tietokantakontti käynnistetään yksinkertaisimmalla tavalla ja luodaan tietokantoja, ne tietokannat häviävät kontin lopetuksen yhteydessä. Tietokantojen häviämistä voitaisiin välttää docker commit –komennon avulla, joka tallentaa käynnissä olevan kontin kaikkine muutoksineen otoksena levykuvaksi. Mutta kontin vikaantumista tai kaatumista ei voida kuitenkaan sulkea pois. Jos kontti kaatuu, kaikki sen sisällä tapahtuneet levymuutokset ja mm. luodut tietokannat, häviävät. Toinen ja varmempi tapa säilyttää kontin sisällä luotuja tietokantoja on docker-osioiden (docker-volume) käyttö. Käytännössä se tarkoittaa sitä, että docker-daemonille ilmoitetaan polku fyysisessä tiedostojärjestelmässä (parametri –v tai --volume), jota kontin annetaan käyttää kirjoitusoikeuksilla. Kontteja käynnistävä komento on silloin muotoa:

docker run –v HostDirectory:ContainerDirectory … imagename,

missä HostDirectory on fyysinen polku isäntäkoneella ja ContainerDirectory polku kansiolle, jonka Docker luo kontin sisälle ja synkronoi em. fyysisen kansion kanssa. Huomautuksena tässä hakemistojen ilmoittamisessa tulee sanoa, että mikäli isäntäkoneen puoleinen, kontille ilmoitettava polku, on kytketty ulkoa, esim. NFS-protokollalla tai iSCSI-tekniikalla, isäntäkoneella on SE Linux aktiivisena ja Docker-palvelu on käynnistetty optiolla --selinux-enabled, kontti ei saa minkäänlaisia oikeuksia kyseiseen kansioon. Ratkaisu tähän ongelmaan on seuraava em. komennon muoto:

docker run –v HostDirectory:ContainerDirectory:Z… imagename

Z-tarkenne sallii Dockerin käyttää kyseistä kansiota. z-tarkenne (pieni z) kertoo Dockerille, että kansiota saa jakaa kaikille konteille. [Docker Cookbook]

7 Yksittäisen kameran käyttöönoton toimintakaava

Kun kaikki olennaiset työn vaiheet ja toimenpiteet on käsitelty, voidaan tehdä yhteenveto yksittäisen kameran käyttöönottoprosessista. Ao. toimenpiteet edellyttävät, että kameratunnusten ja IP-osoitteiden määritysstrategiat on suunniteltu valmiiksi. Kameroiden kanssa

samassa lähiverkossa tulee olla tietokone, jossa Docker on valmiiksi asennettuna. Seuraavassa on kuvattu kohteen ensimmäisen kameran käyttöönottoaskeleet, kun kyseessä on paikallinen, ei pilvessä tapahtuva rekisterinumeroiden tunnistus:

- tietokantakontin käynnistys ja tietokannan määrittely
- kameran konfigurointi
- tiedostojen openalpr.conf ja alprd.conf luonti kamerakohtaiseen kansioon
- kamerakontin käynnistys

7.1 Tietokantakontin käynnistys ja tietokannan määrittely

Mikäli kyseessä on vasta ensimmäinen yrityksen tai kohteen käyttöönotettava kamera, täytyy luoda yrityksen/kohteen oma tietokanta. Ensin fyysiseen tiedostojärjestelmään luodaan kansio, jonka tarkoituksena on säilyttää tietokantaa, konttien sulkemisesta tai kaatumisesta riippumatta:

sudo mkdir /db

, sen jälkeen voidaan käynnistää tietokantakontti:

sudo docker run -d -v /db:/var/lib/mysql -p 3306:3306 --name mysql-s -e MYSQL_ONETIME_PASSWORD=1 -e MYSQL_ROOT_HOST="%" mysql/mysql-server

selvitetään kertakäyttöinen salasana, että voitaisiin päästä tietokannan sisälle:

sudo docker logs mysql-server

tämä komento tulostaa näytölle tietokantakontin käynnistyksen yhteydessä tulleet loki- ja muut ilmoitukset, joissa näkyy muun muassa tietokannan pääkäyttäjän kertakäyttöinen salasana. Seuraavaksi kirjaudutaan tietokantaan:

sudo docker exec -it mysql-s mysql -uroot –p

, vaihdetaan pääkäyttäjän salasana:

ALTER USER 'root'@'localhost' IDENTIFIED BY 'newr00tpassword';

, luodaan käyttäjät kameroille:

CREATE USER 'videouser001'@'%' IDENTIFIED BY 'p455w0rd2';

, luodaan yrityksen tietokanta, johon luodaan yrityksen toimipistettä vastaava taulukko:

CREATE DATABASE COMP_001;

USE COMP_001;

, luodaan yrityksen toimipisteen taulukko:

CREATE TABLE SITE_001 (camera VARCHAR(16), plate VARCHAR(7), dt DATE-TIME);

ja annetaan kameroita vastaaville käyttäjille kirjoitusoikeudet taulukkoon:

GRANT INSERT ON COMP_001.SITE_001 TO 'videouser001'@'%';

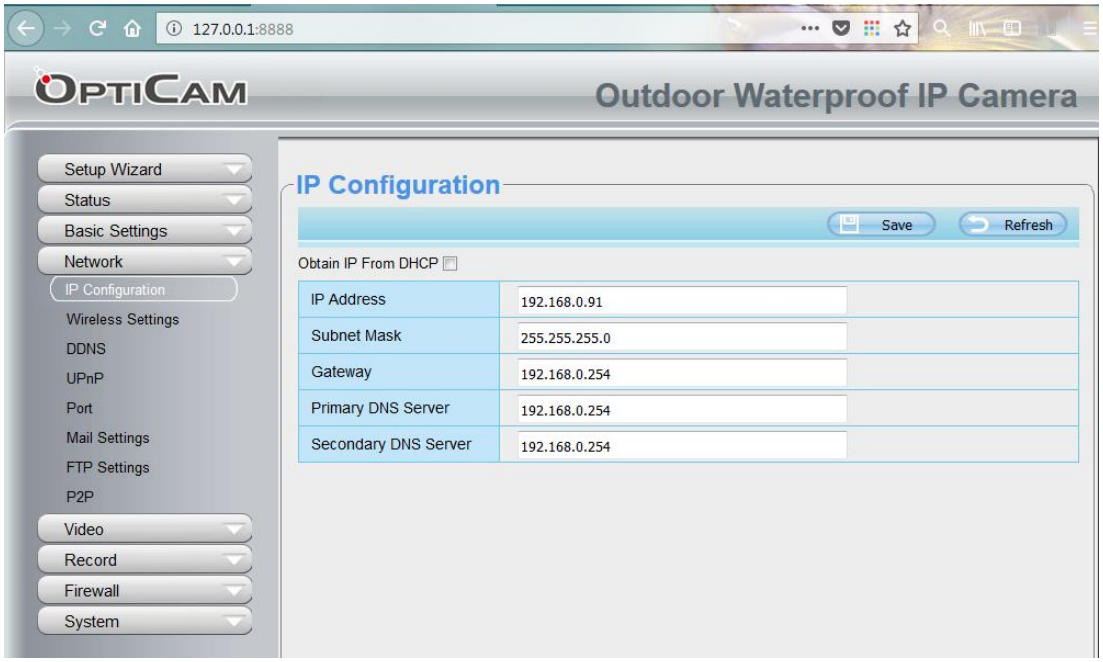
Kun tietokanta, taulukot ja kamerakäyttäjät oikeuksineen on määritelty, tietokantakontin käynnistyksen jälkeen niitä ei enää tarvitse määritellä uudestaan, koska tietokannan tiedostot pysyvät kansiossa **/db** myös kontin lopetuksen jälkeen:

sudo docker start mysql-s

7.2 Kameran konfigurointi

Tyypillinen markkinoilla oleva IP-kamera ensimmäisen käynnistyksen aikana yrittää hakea itselleen dynaamisen IP-osoitteen DHCP-palvelimelta ja jos DHCP-palvelin ei vastaa pyyntöihin, kamera määrittää omaksi IP-osoitteeksi tietyn kiinteän, tyypillisesti 192.168.-alkuisen osoitteen, joka on aina sama kaikissa saman valmistussarjan kame-

roissa. Silloin helpoin tapa määrittää kameraa on kytkeä ethernet-johto kameralta suoraan tietokoneen verkkokorttiin ja asettaa sen verkkokortin osoitteeksi jokin kameran osoitteen kanssa samaan verkkoon kuuluva IP-osoite. Asetusten määritys tapahtuu yleensä selaimen kautta, jolloin graafisen käyttöjärjestelmän käyttö on suositeltavaa, miltei ainoa mahdollinen vaihtoehto. Olennaisin asetus on kameran IP-osoite. Sen tulee kuulua samaan verkkoon konttien hallintatietokoneen kanssa. Kehysnopeudeksi valittiin usean kokeilun jälkeen 20 ja kuvakooksi 640x480.



Kuva 10. Kameran konfigurointisivu. Yhteys kameraan nostettu kuvan kaappauksen hetkellä työpaikalta, ssh-tunnelin kautta, siksi ositekentässä näkyvä osoite on 127.0.0.1

Varsinaisen konfiguroinnin lisäksi kameran fyysinen asento, asennuskohta ja käyttöpaikan valaistus ovat hyvin tärkeitä asioita. Jopa kirkkaan auringonvalon aikaan voidaan tarvita yllättävän paljon lisävalaistusta, kun autot liikkuvat suhteellisen nopeasti [Lighting]. Kameran tulee olla mahdollisimman kohtisuorassa kuvattavaan rekisterikilpeen nähden.

7.3 Tiedostojen openalpr.conf ja alprd.conf luonti

Jokainen tunnistuskontti tarvitsee kahta omaa asetustiedostoa, jotka ovat openalpr.conf ja alprd.conf . Toisella määritetään tunnistusprosessin yksityiskohdat, kuten esim. suurin kameran kuvaamissuunnan ja kuvattavan rekisterikilven välinen kulma.

Toisella asetustiedostolla määritetään daemonin asetukset, kuten tietokanta-asetukset ja kameroiden IP-osoitteet (Liite 2). Nämä kaksi asetustiedostoa luodaan kamerakohtaiseen kansioon, esim.:

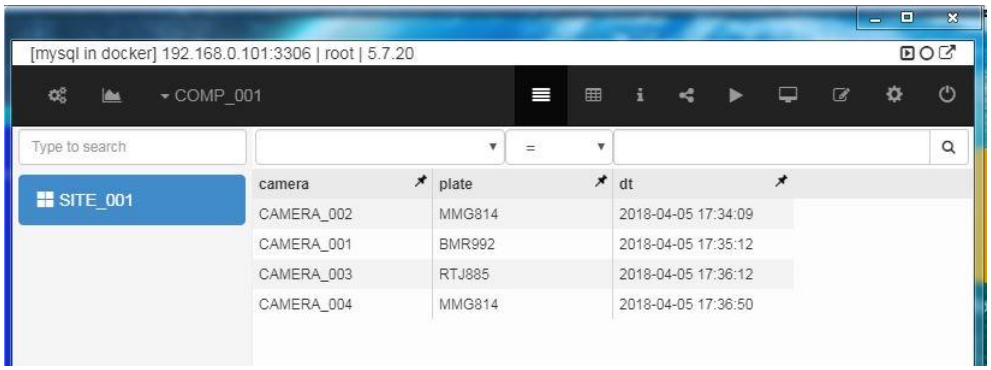
```
cd /kamerat
mkdir c001s001c001; cd c001s001c001
cp /usr/share/openalpr/openalpr.conf.default ./openalpr.conf
nano openalpr.conf
….
cp /usr/share/openalpr/alprd.conf.default ./alprd.conf
nano alprd.conf
….
```

7.4 Kamerakontin käynnistys

docker run -d -v /alpr_conf/c001s001c001:/etc/openalpr/ --name c001s001c001 alprd

8 Tapahtumien selaaminen

Että voitiin selata ALPR-agentin tietokantaan tallennettuja tapahtumia, käytettiin mysql-komentoa ja Chrome-selaimeen asennettua Chrome MySQL Admin –lisäosaa (©Yoichiro Tanaka 2014-2017), lyhennettynä MyAdmin, kun työn tarkoitukseen ei kuulunut erillisen tapahtumien selaustyökalun suunnittelu. MyAdminissa voidaan nostaa yhteys tietokantaan myös SSH-tunnelin kautta, jolloin ei tarvitse nostaa tunnelia erillisen esim. putty-ohjelman avulla. Agentin tietokanta-toiminnon hyödyntäminen reaalimaailmassa edellyttää kuitenkin sovellusta, jonka avulla voitaisiin käsitellä tunnistustapahtumia. Tarvittavan sovelluksen luonne riippuu agentin käyttötarkoituksesta, joka voi olla esimerkiksi puomin nosto parkkipaikalla tai automaattinen laskutus maksullisen tien käytöstä.



Kuva 11. Tunnistustapahtumien selaaminen MyAdminissa

9 Yhteenveto

Tässä insinöörityössä perehdyttiin Docker- ja OpenALPR-tekniikkoihin. Saatiin toimimaan yhdessä OpenALPR, tietokanta MySQL ja Docker. Työn aikana hankittiin kaksi tietokonetta, komponentteja, 4 ulkokäyttöön soveltuvaa kameraa. Tietokoneisiin asennettiin Linux-käyttöjärjestelmät Fedora ja Ubuntu. Tunnistusta toteuttava kokoonpano tehtiin ensin virtuaalisoimattomana ja sitten Dockerissa. Todettiin muun muassa, että Intelin verkkokortti ei toimi AMD:n emolevyssä, minkä takia jouduttiin vaihtamaan AMD-emolevy ja siinä ollut AMD-prosessori Intel-yhteensopivaan emolevyyn ja Intel-prosessoriin. Tehdyn työn tuloksen perusteella voidaan sanoa, että automaattinen rekisterikilpitunnistus voidaan suhteellisen helposti toteuttaa virtuaalisoituna Docker-teknologialla. Jatkokehityksenä tähän projektiin voisi ajatella jonkin Dockerin orkestrointityökalun (esim. Swarm tai Docker-compose) käyttöönottoa. Orkestrointityökalu helpottaa konttien käsittelyä silloin kun kontteja on paljon ja käytetään useampaa tietokonetta konttien ajamiseen. Tunnistuksen tehokkuutta voidaan merkittävästi parantaa hyödyntämällä NVIDIA-näyttökorttien tehoa, jolloin pystytään käsittelemään useita videokehyksiä samaan aikaan. Jälleen jatkokehityksen kohteena voisi olla erillinen selaustyökalu, joka olisi tarkoitettu tallennettujen tunnistustapahtumien käsittelyyn.

Lähteet

[ST 664.10] ST-KORTTI 664.10: ”KAMERAVALVONTAJÄRJESTELMÄT. TEKNINEN SUUNNITTELUOHJE”, Julkaisija: Sähkötieto ry

[Nyrkkisääntö] ”http://doc.openalpr.com/on_premises.html#openalpr-agent” Verkkolähde
[Lighting] “http://doc.openalpr.com/camera_placement.html#lighting” Verkkolähde

[Up and running] ”Docker: Up & Running”, Karl Matthias & Sean P. Kane

[Using Docker] ”Using Docker”, Adrian Mouat, 2016

[Docker Cookbook] ”Docker Cookbook. 80 hands-on recipes to efficiently work with the Docker 1.6”, Neependra Khare, 2015

Insinöörityössä käytetty laitteisto

- Kamerat, 4 kpl:

Opticam O3, System Firmware Version 1.9.1.12, Application Firmware Version 2.54.1.38

- Kytkin:

NetGear 5 ports 10/100/1000

- PC:t, 2 kpl.

Komentojen cat /etc/os-release, uname, lspci, lscpu, lsblk, hwinfo -smartctl ja cat /proc/meminfo tulostukset lyhennetyinä:

Docker_host:

Fedora 23 (Server Edition)
Linux docker_host 4.8.13-100.fc23.x86_64 #1 SMP Fri Dec 9 14:51:40 UTC 2016 x86_64 x86_64 x86_64 GNU/Linux

00:19.0 Ethernet controller: Intel Corporation 82567V-2 Gigabit Network Connection
00:1a.0 USB controller: Intel Corporation 82801JI (ICH10 Family) USB UHCI Controller #4
00:1f.2 SATA controller: Intel Corporation 82801JI (ICH10 Family) SATA AHCI Controller
00:1f.3 SMBus: Intel Corporation 82801JI (ICH10 Family) SMBus Controller
01:00.0 Ethernet controller: Intel Corporation 82574L Gigabit Network Connection

Model name: Intel(R) Core(TM)2 CPU 6400 @ 2.13GHz
CPU MHz: 2133.257

```
sdb 8:16 0 76.7G 0 disk
├─sdb2 8:18 0 36.7G 0 part
│ ├─DockerVolumeGroup-docker--pool_tdata 253:1 0 14.7G 0 lvm
│ │ └─DockerVolumeGroup-docker--pool 253:2 0 14.7G 0 lvm
│ │ └─docker-8:2-122770-7d327c0005aa774c10c584a671decd82dde0e9b4642516c36c...
│ │ 253:3 0 100G 0 dm /var/lib/docker/...
│ └─DockerVolumeGroup-docker--pool_tmeta 253:0 0 40M 0 lvm
│ └─DockerVolumeGroup-docker--pool 253:2 0 14.7G 0 lvm
│ └─docker-8:2-122770-7d327c0005aa774c10c584a671decd82dde0e9b4642516c36c...
│ 253:3 0 100G 0 dm /var/lib/docker/...
└─sdb1 8:17 0 40G 0 part
```

Model Family: Hitachi Deskstar 7K80

Device Model: HDS728080PLA380
sda 8:0 0 111.8G 0 disk
├─sda2 8:2 0 107.8G 0 part /
└─sda1 8:1 0 4G 0 part [SWAP]

Model Family: SandForce Driven SSDs
Device Model: KINGSTON SH103S3120G

MemTotal: 4011488 kB
==========

Ubuntu PC:

16.04.3 LTS (Xenial Xerus)

Linux ubuntu 4.4.0-87-generic #110-Ubuntu SMP Tue Jul 18 12:55:35 UTC 2017 x86_64 x86_64 x86_64 GNU/Linux

00:19.0 Ethernet controller: Intel Corporation 82566DC Gigabit Network Connection (rev 02)
00:1c.0 PCI bridge: Intel Corporation 82801H (ICH8 Family) PCI Express Port 1 (rev 02)
00:1f.2 IDE interface: Intel Corporation 82801H (ICH8 Family) 4 port SATA Controller [IDE mode] (rev 02)
00:1f.3 SMBus: Intel Corporation 82801H (ICH8 Family) SMBus Controller (rev 02)
00:1f.5 IDE interface: Intel Corporation 82801HR/HO/HH (ICH8R/DO/DH) 2 port SATA Controller [IDE mode] (rev 02)
03:00.0 IDE interface: JMicron Technology Corp. JMB368 IDE controller
04:05.0 FireWire (IEEE 1394): LSI Corporation FW322/323 [TrueFire] 1394a Controller (rev 70)

Model name: Intel(R) Core(TM)2 Duo CPU E7200 @ 2.53GHz
CPU MHz: 2527.000

sda 8:0 0 298.1G 0 disk
├─sda1 8:1 0 7.5G 0 part [SWAP]
└─sda2 8:2 0 37.3G 0 part /

Model: "ST3320613AS"

MemTotal: 2037660 kB

Tiedostoon daemon.cpp tehdyt muutokset

```
. . .
//======= added by Valeri Bart
#include <cppconn/driver.h>
#include <cppconn/exception.h>
#include <cppconn/resultset.h>
#include <cppconn/statement.h>
//============================
. . .
struct CaptureThreadData
{
  std::string company_id;
  std::string stream_url;
  std::string site_id;
  int camera_id;
  bool clock_on;
  std::string config_file;
  std::string country_code;
  std::string pattern;
  bool output_images;
  std::string output_image_folder;
  int top_n;
//======= added by Valeri Bart
  std::string DB_addr;
  std::string DB_user;
  std::string DB_pwd;
  std::string DB_cameraname;
//============================
};
. . .
for (int i = 0; i < daemon_config.stream_urls.size(); i++)  {
    pid = fork();
    if (pid == (pid_t) 0)
    {                   // This is the child process, kick off the capture data and upload threads
      CaptureThreadData* tdata = new CaptureThreadData();
      tdata->stream_url = daemon_config.stream_urls[0];
      tdata->camera_id = 1;
      tdata->config_file = openAlprConfigFile;
      tdata->output_images = daemon_config.storePlates;
      tdata->output_image_folder = daemon_config.imageFolder;
      tdata->country_code = daemon_config.country;
      tdata->company_id = daemon_config.company_id;
      tdata->site_id = daemon_config.site_id;
      tdata->top_n = daemon_config.topn;
      tdata->pattern = daemon_config.pattern;
      tdata->clock_on = clockOn;
        //======= added by Valeri Bart
            tdata->DB_addr = daemon_config.db_addr;
            tdata->DB_user = daemon_config.db_user;
            tdata->DB_pwd = daemon_config.db_pwd;
            tdata->DB_cameraname = daemon_config.db_cameraname;
        //==================================
   tthread::thread* thread_recognize = new tthread::thread(streamRecognitionThread, (void*) tdata);
      if (daemon_config.uploadData)
. . .
       //======= added by Valeri Bart
       try {
            DBdriver = get_driver_instance();
            DBcon = DBdriver->connect( tdata->DB_addr, tdata->DB_user, tdata->DB_pwd);
            DBcon->setSchema(tdata->company_id);
            DBstmt = DBcon->createStatement();
            DBres = DBstmt->executeQuery("INSERT INTO "+tdata->site_id+\
                        "(camera, plate, dt) VALUES ('"+tdata->DB_cameraname+"','"+\
                       results.plates[0].bestPlate.characters+"', now());");
```

```
            while ( DBres->next()) {
                LOG4CPLUS_WARN(logger,"request: " << DBres->getString(1) << std::endl);
            }
  } catch (sql::SQLException &e) {
      if(e.getErrorCode() != 0 ){
          LOG4CPLUS_WARN(logger, "# ERR: SQLException in " << __FILE__);
          LOG4CPLUS_WARN(logger, "(" << __FUNCTION__ << ") on line " << __LINE__ <<
std::endl);
                LOG4CPLUS_WARN(logger, "# ERR: " << e.what());
                LOG4CPLUS_WARN(logger, " (MySQL error code: " << e.getErrorCode());
                LOG4CPLUS_WARN(logger,", SQLState: " << e.getSQLState() << std::endl);
            } else {
                LOG4CPLUS_WARN(logger,"Event succefully saved..." << e.getSQLState() <<
std::endl);
            }
      }
      delete DBres;
      delete DBstmt;
      delete DBcon;
. . .
```