
Eero Tuure

# PKI-varmentajan rakentaminen osaksi testiautomaatiota

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

5.8.2021

# Tiivistelmä


| | |
|---|---|
| Tekijä: | Eero Tuure |
| Otsikko: | PKI-varmentajan rakentaminen osaksi testiautomaatiota |
| Sivumäärä: | 49 sivua |
| Aika: | 5.8.2021 |
| Tutkinto: | Insinööri (AMK) |
| Tutkinto-ohjelma: | Tieto- ja viestintätekniikka |
| Ammatillinen pääaine: | Ohjelmistotuotanto |
| Ohjaajat: | Squad Group Leader Osmo Kaukanen<br>lehtori Simo Silander |


---


Insinöörityössä tutkittiin PKI (Public Key Infrastucture) -rakennetta, -menetelmiä ja -standardeja. Työn tarkoituksena oli selvittää, mitä vaatisi rakentaa CMP (Certificate Management Protocol) -protokollaa tukeva CA (Certificate Authority) -palvelin, jonka tarkoituksena olisi toimia ulkoisen CMP-palvelimen sijaisena testitapausten suorituksissa.

Työssä selvitettiin, mitä avoimen lähdekoodin PKI-ohjelmistoja on saatavilla, sekä tutkittiin kohdesuoritusympäristönä toimivan Kubernetes-hallintaohjelmiston rakennetta ja mahdollisuuksia. Selvityksen perusteella insinöörityössä suunniteltiin EJBCA-ohjelmistosta ja muista tarpeellisista palveluista CA-kokonaisuus, joka tukisi CMP- sekä LDAP-protokollia ja jota voitaisiin suorittaa Kubernetes-klusteriympäristössä.

Insinöörityössä tutkittiin EJBCA-ohjelmiston mahdollisuuksia ja konfigurointia sekä perehdyttiin syvemmin CMP-protokollaviestien rakenteisiin. Työssä keskityttiin 3GPP-standardissa määriteltyyn CMPv2-profiiliin, joka määrittää tarkemmat rajoitukset CMP-protokollaan. Tämän tarkastelun pohjalta selvitettiin, minkälaiset CMP-määritykset ovat mahdollisia ja miten tehdyt määritykset olisivat aina valmiina määriteltyjä asennettaessa ohjelmisto Kubernetes-klusteriympäristöön.

Työssä suunniteltiin, kuinka kehitetystä palvelusta saataisiin helposti uudelleen määriteltävä testitapauksia varten. Palvelun valmiin konfiguraation toteutuksessa ilmeni myös haasteita, kuten että kaikkia konfiguraatioita ei pystytty ohjelmiston komentotulkin kautta asettamaan. Tähän vaadittiin muita lähestymistapoja. Lisäksi konfiguraatioiden suorittaminen hidastaa palvelukokonaisuuden käynnistymistä Kubernetes-ympäristössä, mihin insinöörityössä pohdittiin ratkaisua.

Lopputuloksena palvelukokonaisuus liitettiin olemassa olevaan CI/CD-järjestelmään korvaamaan ulkoinen CMP-palvelin testitapausten suorittamisessa. Työn tuloksia ja tehtyä selvitystyötä voidaan hyödyntää CA-palveluiden rakentamisessa, etenkin Kubernetes-klusteriympäristössä. Insinöörityö antaa myös tarkempaa kuvausta CMP-protokollasta ja sen operaatioista.



Avainsanat: PKI, Kubernetes, testiautomaatio, CMP

# Abstract


| | |
|---|---|
| Author: | Eero Tuure |
| Title: | Building PKI Certificate Authority into Test Automation |
| Number of Pages: | 49 pages |
| Date: | 5 August 2021 |
| Degree: | Bachelor of Engineering |
| Degree Programme: | Information and Communication Technology |
| Professional Major: | Software Engineering |
| Supervisors: | Osmo Kaukanen, Squad Group Leader |
| | Simo Silander, Senior Lecturer |


---


The purpose of the study was to research the PKI structure and protocols and implement a substitute for an external CA (Certificate Authority) to reduce test case failures of CMP (Certificate Management Protocol) operations.

The study investigated the EJBCA open source PKI software, and the structure and possibilities of the Kubernetes management software from the test automation point of view. The possibilities and configuration of EJBCA software were studied, and the structures of CMP-protocol messages were studied in more depth. The investigation focused on the CMPv2-profile defined in the 3GPP-standard, which specifies the more specific limitation of the CMP-protocol.

The thesis goes through possibilities for CMP-configurations to be pre-defined when installing EJBCA into the Kubernetes cluster, and ways to make EJBCA easy to configure for the new test cases. There were challenges in implementing a ready-made configuration of the service, such as not being able to set all configurations via Unix Shell requiring different approaches. In addition, the execution of the configurations slows down the start-up time of the server in the Kubernetes, for which a solution was searched.

As a result, the EJBCA was integrated into the existing CI/CD system to replace the external CA in E2E (End to End) -test case executions. The results and the research can be utilized in the construction of CA services, especially in the Kubernetes environment. The thesis also gives a more detailed description of some CMP-protocol messages and operations.



Keywords: PKI, Kubernetes, test automation, CMP

# Sisällys

## Lyhenteet

PKI: *Public Key Infrastructure*. Julkisten avainten hallintaan tarkoitettu rakenne.

CA: *Certificate Authority*. PKI:ssa toimiva luotettu sertifiointiviranomainen, joka vastuullisesti myöntää varmenteita loppukäyttäjille.

CMP: *Certificate Management Protocol*. X.509-standardin digitaalistenvarmenteiden hallintaan tarkoitettu Internet-protokolla.

E2E: *End to End*. Testausmenetelmä, jossa testataan järjestelmää kokonaisuutena.

CRL: *Certificate Revocation List.* Luettelo digitaalisista varmenteista, jotka myöntävä sertifiointiviranomainen on mitätöinyt ennen niiden määräaikaa ja joihin ei pitäisi enää luottaa.

LDAP: *Lightweight Directory Access Protocol*. Hakemistopalveluiden hallintaan tarkoitettu Internet-protokolla.

CI: *Continuous Integration*. Jatkuva integraatio, ohjelmistotuotannon menetelmä, jossa useiden tekijöiden muutokset yhdistetään ohjelmistoprojektiin ja suoritetaan automaattiset koonnit ja testit.

EJBCA: Insinöörityössä käytetty PKI-ohjelmisto.

# 1 Johdanto

Insinöörityössä on tarkoituksena tutkia X.509-standardin julkisen avaimen infrastruktuuria, protokollia ja haasteita. Tutkimuksen lisäksi insinöörityön tavoitteena on ottaa käyttöön sertifikaattihallintamikropalvelun testiautomaatioympäristössä varmentaja (Certificate Authority) käyttäen CMP (Certificate Management Protocol) -protokollaa HTTP-protokollan päällä. Toteutettavan palvelun tarkoitus on helpottaa erilaisia CMP-operaatioiden E2E (End to End) -testejä poissulkemalla yhteysongelmat ulkopuoliseen varmentajaan, mitkä viivästyttävät toteutuksien valmistumista aiheuttamalla ylläpitoa ja testitapausten suorittamisen epäonnistumista. Työn tilaajana on maailmanlaajuisesti toimiva tietoliikennealan yhtiö Nokia Oyj. Aihe valikoitui oman kiinnostuksen ja työnantajalle tuovan hyödyn kautta.

Nykyään monet organisaatiot turvautuvat julkisen avaimen infrastruktuuriin (PKI, Public Key Infrastructure) ylläpitääkseen salauksen turvallisuutta. Tarkemmin sanoen yleisimpään salausmenetelmään tänä päivänä liittyy julkinen avain, jota kuka vain voi käyttää viestin salaukseen, ja yksityinen avain, jota vain yhden henkilön pitäisi käyttää viestin purkamiseen. Näitä avaimia voivat käyttää ihmiset, laitteet ja ohjelmistot. PKI-tietoturva ilmestyi julkisuuteen 1990-luvulla auttaakseen hallitsemaan salausavaimia digitaalisten varmenteiden myöntämisen ja hallitsemisen kautta. Näiden varmenteiden tarkoituksena on varmistaa yksityisen avaimen ja omistajan suhteen aitous turvallisuuden ylläpitämiseksi. Varmenteet ovat periaatteena samanlaisia kuin ajokortti tai passi. (Keyfactor – What is PKI.)

Insinöörityössä keskitytään X.509-standardin digitaalisiin varmenteisiin ja protokolliin sekä varmentajien toimintaan testausautomaation näkökulmasta. Insinöörityöraportissa käsitellään eri aihealueita muodostamalla lukijalle ymmärrys insinöörityön vaiheista. Raportin alussa käydään läpi PKI:n olennaisia konsepteja ja rakennetta. Seuraavaksi käsitellään CA (Certificate Authoirty) -palvelun suunnittelua, haasteita ja toteutusta Kubernetes-ympäristössä. Lopuksi raportoidaan,

kuinka varmentaja liitetään osaksi sertifikaattipalvelun testiautomaatiota ja käsitellään siihen liittyviä haasteita.

# 2 Julkisen avaimen infrastruktuuri

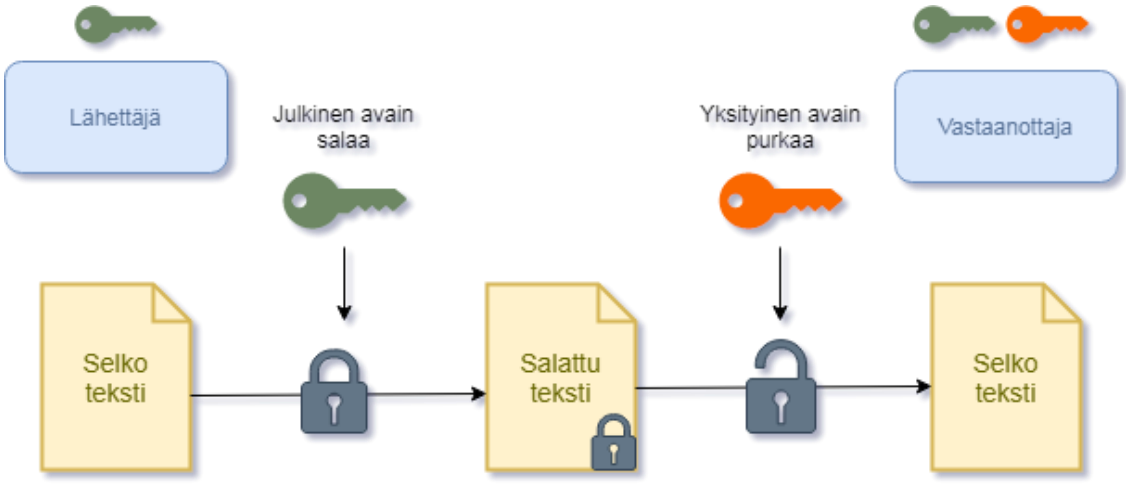
## 2.1 Tarkoitus ja rakenne

Jotta turvallisuusprotokollat kuten Secure Multipurpose Internet Mail Extensions (S/MIME), Transport Layer Security (TLS) ja Internet Protocol Security (IPSec) toimisivat tehokkaasti, pitää pystyä mahdollistamaan palveluiden todentaminen, kiistämättömyys ja luottamuksellisuus. Nämä käsitteet ovat turvallisuuden kolme perusvaatimusta. Luottamuksella tarkoitetaan tiedon salassapitoa, todentamisella identiteettien todentamista ja kiistattomuudella tiedon turvallisen välittämisen varmistamista. (Ristić 2014: 4.)

Edellä mainitut protokollat turvautuvat symmetriseen ja epäsymmetrisen salaukseen sekä tiedon turvalliseen välittämiseen (Xenitellis 2000: 30). Symmetrisellä salauksella tarkoitetaan salausmenetelmää, jossa tieto voidaan salata ja purkaa samalla salausavaimella. Salausavain on salausalgoritmilla luotu bittijono, mitä käytetään tiedon hajottamiseen niin, ettei tiedon sisällöstä saada enää selkoa. Salausalgoritmeja on useita, joista nykyään käytetyimmät ovat AES (Advanced Encryption Standard) symmetriseen salaukseen ja RSA (Rivest–Shamir–Adlema) epäsymmetriseen salaukseen. (Ristić 2014: 5-13.) Symmetrisen luonteen takia ei symmetristä salausavainta voida kovin turvallisesti jakaa. Jos symmetrisen salausavaimen jakeluun käytetty kanava vaarantuu, koko suojattujen viestien järjestelmä ei ole enää turvallinen. (Keyfactor – What is PKI.)

Toinen mainituissa turvallisuusprotokollissa käytetty salausmenetelmä on epäsymmetrinen salaus, eli julkisen avaimen salaus. Toisin kuin symmetrisessä salauksessa, missä samalla avaimella salataan ja puretaan, epäsymmetrisessä

salauksessa käytetään avainparia. Avainpari sisältää yhden muiden käyttöön jaettavan julkisen avaimen ja itselle salassa pidettäväksi jäävän yksityisen avaimen. Kuten epäsymmetrisen salauksen nimestä voidaan päätellä, samalla avaimella ei voida purkaa sillä salattua tietoa, vaan tarvitaan toinen avainparista sen purkamiseen (Keyfactor – What is PKI). Julkisella avaimella salatun tiedon purkaminen vaatii yksityisen avaimen ja yksityisellä salattu julkisen avaimen.




Kuva 1. Epäsymmetrinen salaaminen julkisella avaimella.

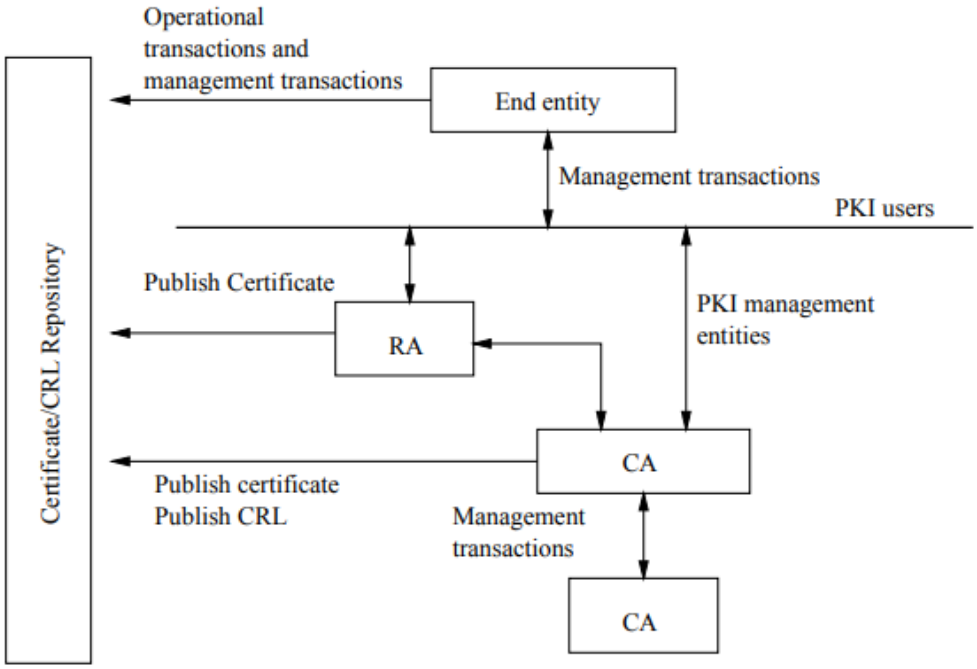
Kuvassa 1 on esitetty epäsymmetrisen salauksen toimintaa, jossa tiedon lähettäjä salaa tekstin vastaanottajalta tarjoamalla julkisella avaimella ja vastaanottaja purkaa sen omalla yksityisellä avaimella. Epäsymmetrisessä salauksessa hyvää on se, ettei kukaan toinen pysty purkamaan itselle tarkoitettuja viestejä, kunhan yksityinen avain pysyy salassa. Jos yksityinen avain jostain syystä pääsee vuotamaan, on avainpari käyttökelvoton. Julkista avainta voi taas huolehtimatta jakaa kenelle vain. Toisaalta epäsymmetriset avaimet ovat kooltaan suurempia kuin symmetriset avaimet ja vaativat enemmän laskennallista tehoa tiedon salaamiseen ja purkamiseen. (Ristić 2014: 12-13.)

Molemmissa salausmenetelmissä on hyviä ja huonoja puolia, mutta näitä yhdistämällä saadaan vahva ja luotettava salausprotokolla. Esimerkiksi TLS-yhteyksissä epäsymmetristä salausta käytetään yhteyden luomiseen lähettämällä julkinen avain, millä vastaanottaja pystyy salaamaan uuden symmetrisen avaimen

molempien käytettäväksi. Näin saadaan turvallisesti välitettyä isomman tietomäärän salaamiseen tehokkaampi symmetrinen avain. Ongelmaksi vielä jää, miten voidaan todentaa julkinen avainpari ja sitä jakavan tahon identiteetti. Tähän ratkaisuna on toimittaa epäsymmetrisen avainparin julkinen avain digitaalisen varmenteen muodossa. [Xenitellis 2000: 18; Ristić 2014: 12.]

Digitaalinen varmenne on käytännössä luotettavan tahon allekirjoittama todistus julkisen salausavaimen luotettavuudesta ja voimassaolosta (Cooper ym. 2008: 8-9). Jotta voidaan mahdollistaa palveluiden luotettava ja tehokas digitaalisten varmenteiden hallinta, tarvitaan julkisen avaimen infrastruktuuria (Ristić 2014: 13-14.)

PKI (Public Key Infrastructure) eli julkisen avaimen infrastruktuuri on joukko laitteistoja, ohjelmistoja, käyttäjiä, käytäntöjä ja menetelmiä, joita tarvitaan digitaalisten varmenteiden luomiseen, ylläpitoon, käyttöön, tallentamiseen ja kumoamiseen (Xenitellis 2000: 34). PKI on myös se, joka sitoo avaimet identiteetteihin varmentajan tarjoamalla rekisteröintiprosessilla ja varmenteen myöntämisellä (Chokani ym. 2003: 3). Kuvassa 2 on esitetty PKI-rakennetta ja sen eri tahojen toimintaa.

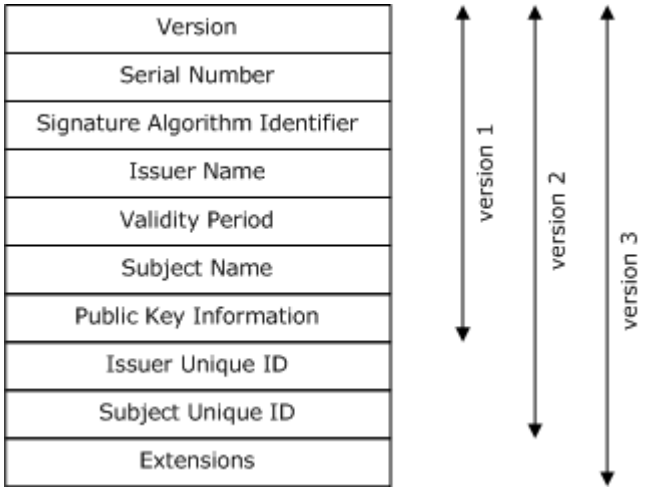



Kuva 2. PKI-entiteetit. [Xenitellis 2000: 35; RFC5280 2008: 8]

Jotta voidaan myöntää ja hylätä varmenteita, tarvitaan varmentaja (CA, Certificate Authority), joka takaa varmenteen oikeellisuuden, sekä rekisteröijä (RA, Registration Authority), joka vastaa varmenteiden pyyntöjen hyväksymisestä ja varmentamisesta. Voimassa olevien varmenteiden tallentamiseen tarvitaan arkisto, josta varmenteita voidaan hakea. Arkistoon myös tallennetaan varmenteiden hylkäyslista (CRL, Certificate Revocation List), jos tällainen on varmentajalla käytössä. Varmenteiden hylkäyslista on julkinen binääriformaatissa oleva dokumentti, joka sisältää listan kaikista varmentajan myöntämistä varmenteista, joihin ei tulisi enää luottaa. Muita tahoja PKI:ssa ovat varmenteiden haltija ja loppukäyttäjä. Varmenteen haltija on tyypillisesti jokin palvelu tai verkkoliikennettä käyttävä laite, jolle varmenne on myönnetty ja jota loppukäyttäjä käyttää. (Xenitellis 2000: 35-36; Ristić 2014: 63-64.)

## 2.2 Varmenteet

Aiemmin mainittiin tarpeesta varmentaa julkinen avainpari ja sitä jakavan palvelun luotettavuus. Julkisen avaimen infrastruktuurissa julkisten avainparien luotettavuus ja niiden omistajien kiistämättömyys tapahtuu digitaalisten varmenteiden kautta. Varmenne on digitaalinen dokumentti, joka sisältää yksinkertaistettuna julkisen avaimen, tietoa avaimen omistajasta, varmenteen myöntäjän digitaalisen allekirjoituksen ja allekirjoituksessa käytetyn algoritmin tunnisteen, ja muuta yleistä tietoa varmenteesta, kuten voimassaoloajan. (Ristić 2014: 66.)



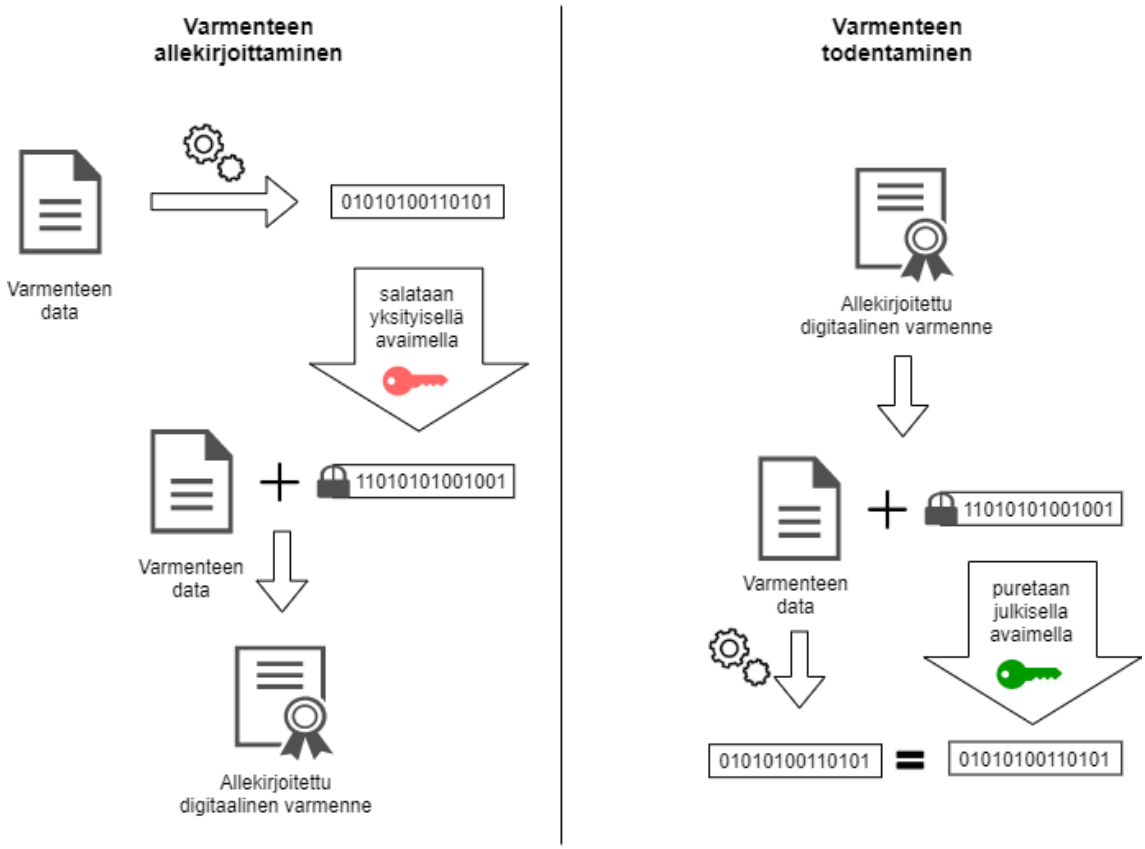

Kuva 3. X.509-standardin julkisen avaimen varmenteen kentät. (Hickey ym. 2017)

Kuten kuvasta 3 nähdään, X.509-standardin varmenne koostuu kentistä ja lisäksi kolmannen version varmenteissa erilaisista laajennuksista. Kentistä ensimmäinen on varmenteen versio, joita on kolme, ja uudempi versio tukee aina vanhempia. Ensimmäinen versio tukee peruskenttiä, toisessa versiossa on lisänä myöntäjän id ja subjekti, eli kohteen id. Kolmas versio tuo mahdollisuuden laajennuksille kuten esimerkiksi varmenteiden hylkäyslistan jakeluosoitteelle. (Ristić 2014: 67-70.)

Tietojen tarkoituksena on todistaa yhdistettävän tahon aitous ja luotettavuus sertifiointiviranomaisen (Certificate Authority) avulla. Sertifiointiviranomainen

myöntää varmenteen ja allekirjoittaa sen omalla yksityisellä avaimellaan. Tämänlaista menetelmää kutsutaan digitaaliseksi allekirjoitukseksi. (Ristić 2014: 66.)

Digitaalinen allekirjoitus on suojausmenetelmä, minkä avulla voidaan varmistaa digitaalisen viestin tai asiakirjan aitous. Epäsymmetrisen salauksen luonne mahdollistaa digitaalisen allekirjoituksen algoritmin luomisen, minkä avulla yksityisen avaimen allekirjoittama viesti voidaan vahvistaa julkisella avaimella. Tarkempi algoritmin lähestymistapa riippuu käytetystä julkisen avaimen algoritmista. Esimerkiksi RSA (Rivest–Shamir–Adleman) -algoritmi, joka on käytetyin julkisen avaimen algoritmi, voidaan yhdistää hajautusfunktioon dokumentin muuttumattomuuden todistamiseksi. (Ristić 2014: 13-14.)




Kuva 4. Digitaalisen varmenteen allekirjoitus ja todentaminen.

Kuvassa 3 on esitetty, miten allekirjoittaja voi käyttää hajautusfunktiota oman sormenjäljen luomiseen ja salaa sen omalla yksityisellä avaimella. Tarkistaja

pystyy allekirjoittajan luotetulla julkisella avaimella purkamaan salatun sormenjäljen ja tarkistamaan allekirjoituksen aitouden käyttämällä samaa hajautusfunktiota. Jos julkisella avaimella purettu hash-bittijono vastaa alkuperäistä, voidaan olla varmoja tiedon muuttumattomuudesta, koska pienikin tiedon muutos tuottaa täysin erilaisen hash-bittijonon. (Ristić 2014: 13-14.)

Yleisin allekirjoituksessa käytetty hajautusfunktio on NIST:n (National Institute of Standards and Technology) ja NSA:n (National Security Agency) kehittämä SHA (Secure Hash Algorithm) (Scheiner 1996: 365). Digitaalisessa allekirjoituksen algoritmin valitsimessa tulee huomioida, että kaikki julkisen avaimen algoritmit eivät tue digitaalista allekirjoitusta saman lailla kuin RSA. Muita käytettyjä julkisen avaimen algoritmeja kuten DSA tai ECDSA ei voida käyttää digitaalisen allekirjoituksen salaamiseen samalla tavalla, vaan vaatii erilaisen lähestymistavan. (Ristić 2014: 252.)

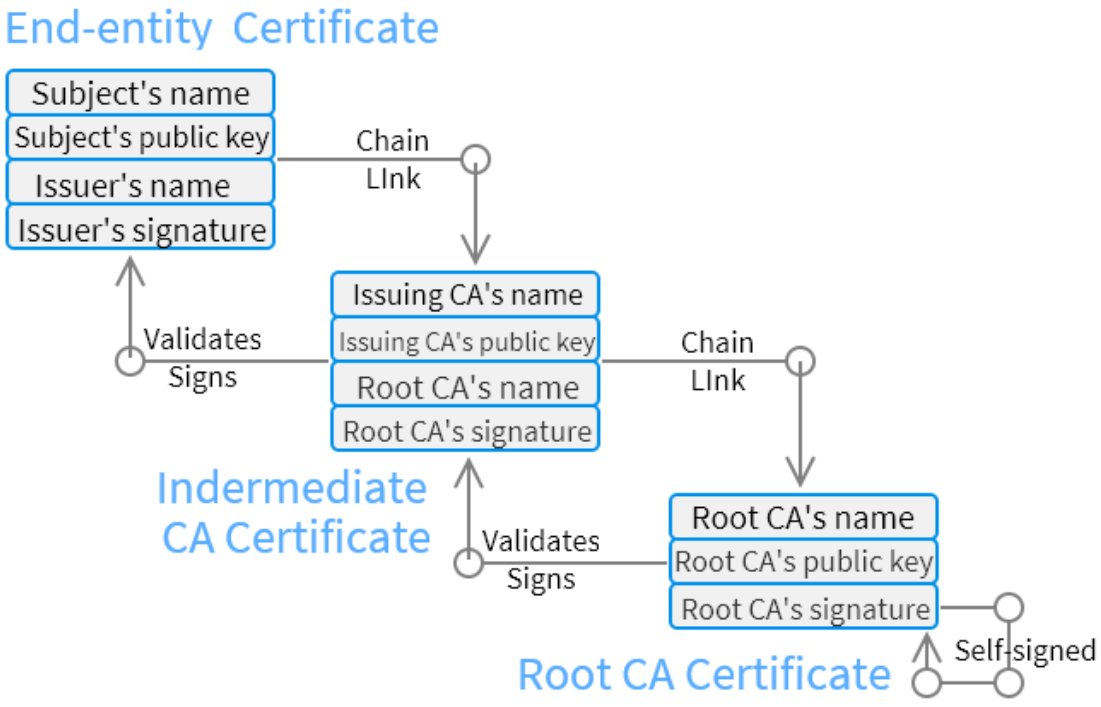
Varmenne voi olla koodattuna erilaisiin ASN.1 (Abstract Syntax Notation One) -standardin binaariformaatteihin, kuten BER:iin (Basic Encoding Rules). X.509-varmenteet turvautuvat DER (Distinguished Encoding Rules) -formaattiin, joka on yksiselitteinen BER-muunnos ja kriittinen salauskäytössä, varsinkin digitaalisessa allekirjoituksessa. Useimmat varmenteet toimitetaan PEM (Privacy-Enhanced Mail) -formaatissa, joka on DER-formaatti muunneltu ASCII-formaattiin, kun käytetään Base64-koodausta. (Ristić 2014: 67.)

## 2.3 Varmentajat

Edellisessä luvussa puhuttiin varmenteista ja siitä, kuinka sertifiointiviranomaisia (Certificate Authority) eli varmentajia tarvitaan varmenteiden myöntämiseen. Varmentajat ovat tärkein osa nykyistä internetin luottamusmallia. Varmentajien ensisijaisena tehtävänä on digitaalisten varmenteiden myöntäminen vastuullisesti sen jälkeen, kun on varmistettu, että pyyntö tulee joltakin, jolla on valtuus tehdä tällainen pyyntö. (Ristić 2014: 74-75.)

Varmentajat voivat myöntää varmenteita mille tahansa valtuutetulle entiteetille kuten palveluille, laitteille, muille varmentajille, tai esimerkiksi itselleen, mikä mahdollistaa oman sertifiointiviranomaisen luomisen omien palveluiden varmentamiseen (Ristić 2014: 74). Itse allekirjoitettua varmennetta kutsutaan juurivarmenteeksi. Haasteena juurivarmentajan luomisessa on, miten muut voivat luottaa siihen. Käyttöjärjestelmät, verkkoselaimet ja muut ohjelmistot, jotka suorittavat verkkopyyntöjä, sisältävät ja ylläpitävät listaa luotetuista juurivarmentajista. Esimerkiksi Windows-järjestelmällä voidaan tarkistella, viedä, tuoda tai poistaa juurivarmenteita ”certmgr.msc”-työkalulla. Luotettuja juurivarmenteita voi siis itse asentaa laitteisiin, mikä mahdollistaa oman varmentajan käytön esimerkiksi organisaation sisäverkossa ilman ylemmän luotetun juurivarmentajan allekirjoitusta.

Koska varmentaja voi myöntää varmenteen myös toiselle varmentajalle, voidaan varmenteita linkittää toisiinsa, mistä syntyy kuvan 5 tapainen varmenteiden luottamisketju (Ristić 2014: 253; Keyfactor – What is PKI).




Kuva 5. Varmenteen luottamusketju (Keyfactor – Certificate Chain of Trust).

Luottamusketju koostuu vähintään kolmesta komponentista: juurivarmentajan varmenteesta, keskiasteen varmentajan varmenteesta ja palvelun varmenteesta. Kuvan 5 tapaisessa luottamusketjussa juurivarmentaja on myöntänyt keskiasteen varmentajalle varmenteen, joka voidaan todistaa varmenteessa olevasta allekirjoituksesta juurivarmentajan julkisella avaimella. Keskiasteen varmentaja on myöntänyt varmenteen loppukäyttäjälle, jonka varmenne tarkistetaan käymällä tämä ketju läpi, kunnes päädytään juurivarmenteeseen.

Varmenteiden ketjuttaminen mahdollistaa vastuun jakamisen muille varmentajille. Tyypillisesti juurivarmentaja myöntää varmenteita vain keskiasteen varmentajalle, joka myöntää varmenteita loppuentiteeteille. (Keyfactor – What is PKI.)

Tämän tapaisessa luottamusketjussa joskus ketjun juurivarmenne vanhenee ja koko luottamisketju katkeaa. Vaikka olisi hienoa, jos jokainen ohjelmisto päivitettäisiin usein ja toimittaisi päivitetyn version juurivarmenteesta, näin ei kuitenkaan aina tehdä useista eri syistä. Ratkaisuna tähän on juurivarmenteiden ristiin allekirjoitus, mikä mahdollistaa usean juurivarmenteen varmentamisen yhdellä varmenteella. Tällöin ei ole tarvetta päivittää jokaista juurivarmennetta laitteeseen erikseen. (Schatten 2020.)

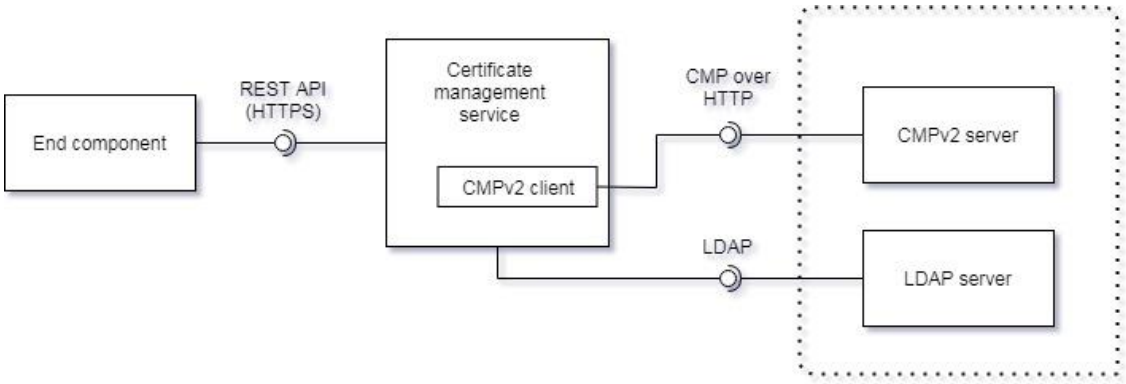
Toinen hyöty ristiin varmentamisesta on tilanteissa, joissa varmentajan yksityinen avain jostain syystä vuotaa. Esimerkiksi oletetaan, että varmentajan ”A” avain vuotaa ja tämän varmentajan sertifikaatin ovat allekirjoittaneet varmentajat ”A” ja ”B”. Vuodon seurauksena varmentaja ”A” mitätöi julkisen avaimensa, eikä voida luottaa enää mihinkään varmentajan ”A” myöntämään varmenteeseen. Koska varmenteen on allekirjoittanut myös varmentaja ”B”, kaikki varmentajaan ”B” luottavat entiteetit voivat silti ylläpitää luottamuksen tason. Toisin sanoen ristiin allekirjoittaminen tuo lisävarmuutta luottamisketjuun. (Schatten 2020.)

# 3 CA-palvelimen suunnittelu

Erilaiset PKI-ohjelmistot, jotka käyttävät ulkoista sertifiointiviranomaista varmenteiden hankkimiseen, ovat riippuvaisia sen toiminnasta. Insinöörityöaihe pohjautui tarpeesta vähentää yhteysongelmia CA (Certificate Authority) -palvelun ja sen toiminnasta riippuvaisen sertifikaattihallintapalvelun E2E (End to End) -testitapausten suorittamisessa. Ongelman ratkaisemiseksi insinöörityössä oli tarpeena tutkia PKI-varmentajien toimintaa ja tulosten perusteella rakentaa virtuaalisessa klusteriympäristössä suoritettava käyttövalmis CA-palvelu, joka olisi helposti määriteltävissä uusia testitapauksia varten.

## 3.1 Tausta ja tavoitteet

Sertifikaattihallintamikropalvelulla tässä kontekstissa tarkoitetaan palvelua, jonka tarkoituksena on hakea ulkopuolisen varmentajan allekirjoittamia sertifikaatteja käyttäen varmenteiden hallintaan tarkoitettua CMP-protokollaa HTTP-protokollan päällä, kuten kuvassa 6 on esitetty. Sertifikaattihallintapalvelu auttaa sen loppukomponenttien varmenteiden hakemista ja päivittämistä.




Kuva 6. Yksinkertaistettu kuvaus sertifikaattihallintapalvelun riippuvuudesta CMP- ja LDAP-palvelimiin.

Loppukomponentilla tarkoitetaan tässä ympäristössä jotain komponenttia, kuten puhelinverkon tukiasema, joka vaatii varmenteen CMP-palvelimena toimivalta

sertifiointiviranomaiselta ulkoisen liikenteen suojaamiseksi ja käyttää sertifikaattihallintapalvelua sen saamiseksi. Tämän sertifikaattihallintapalvelun CMP-operaatioiden E2E-testeissä käytetään toistaiseksi ulkoista CMP-palvelinta, joka ei ole sertifikaattihallintapalvelun kehitystiimin hallinnoima. Koska CMP-palvelin sijaitsee testiympäristön ulkopuolella, aiheuttaa se useita yhteys- /palomuurion-gelmia. Lisäksi koska CMP-palvelin ei ole kehitystiimin hallinnoima, joudutaan kaikki ongelmatilanteet käsittelemään omanlaisensa prosessin kautta, mikä lisää korjausten vasteaikaa. Tähän ongelmaan tulisi löytää ratkaisu, joka eliminoisi E2E-testisuoritusten epäonnistumisen ulkoisten yhteyksien takia.

Koska CA-ohjelmisto tulisi testikäyttöön, olisi se rahoituksen puolesta hyvä olla avoimen lähdekoodin CMPv2-protokollaa tukeva toteutus ja näin toimisi CMP-palvelimena sertifikaattihallintapalvelulle. Lisäksi sertifikaattihallintapalvelu tukee vain LDAP (Lightweight Directory Access Protocol) -hakemistoprotokollaa varmenteiden hylkäyslistan hakemista varten. Tämän takia tulisi toteutettavan CA-palvelun järjestää LDAP-implementaatio, minne hylkäyslistat julkaistaan.

CA-palvelun konfiguraatiot, kuten X.509-standardin sertifikaattiprofiilit, tulisi myös olla mahdollisimman kätevää ja helppoa. Kaiken kaikkiaan CMP-palvelimesta haluttaisiin helposti käyttöönotettava klusterissa ajettava toteutus, joka on helposti uudelleenmääriteltävissä uusia testitapauksia varten.

## 3.2 Suoritusympäristö

Koska sertifikaattihallintamikropalvelun sekä muiden sen sovelluskomponenttien E2E-testit suoritetaan konttiperustaisessa Kubernetes-klusteriympäristössä, missä käytetään Helm-pakettihallintajärjestelmää, tarvitsee ymmärtää näiden toimintaa ja ominaisuuksia.

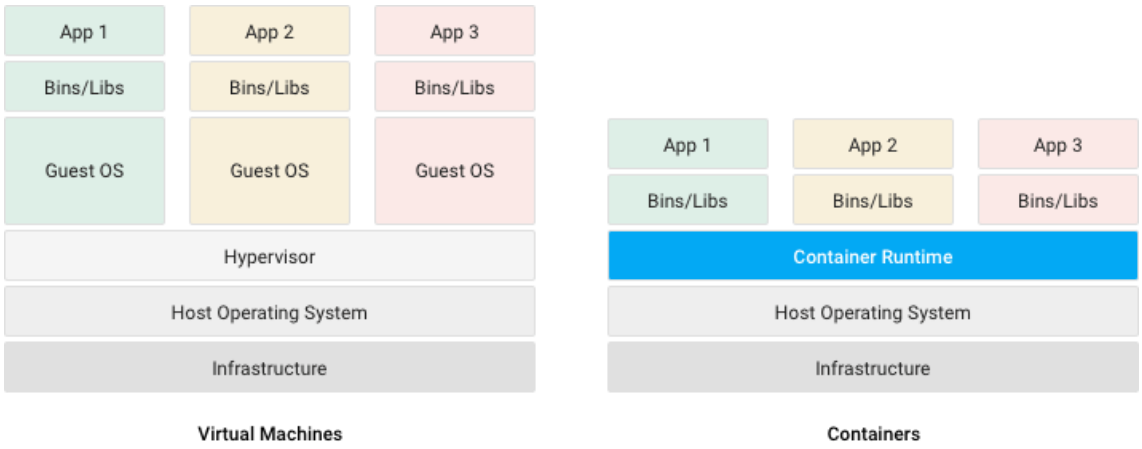
Kubernetes tai lyhenteeltään tunnettu K8s on Googlen alun perin suunnittelema, ja nykyään CNCF:n (Cloud Native Computing Foundation) ylläpitämä avoimen lähdekoodin hallintaohjelmisto Docker-sovelluskonttien käyttöönoton,

skaalauksen ja hallinnan automatisointiin (What Is Kubernetes 2021). Jotta saadaan käsitys, mitä Kubernetes ja Helm käytännössä tarjoavat, tarvitsee ensin tutkia, mitä on käyttöjärjestelmätason virtualisointi ja Docker-ohjelmisto.

### 3.2.1 Käyttöjärjestelmätason virtualisointi

Käyttöjärjestelmätason virtualisointi tai yksinkertaistettuna säiliöinti on looginen pakkausmekanismi, jossa sovellukset voidaan erottaa ympäristöistä, missä ne oikeasti pyörivät. Tämän erottamisen ansiosta konttipohjaiset sovelluksen voidaan ottaa käyttöön helposti ja johdonmukaisesti riippumatta siitä, onko kohdeympäristö yksityinen datakeskus, julkinen pilviympäristö tai kehittäjän henkilökohtainen tietokone. Säiliöinti tarjoaa selkeän edun kehitysprosessiin, koska kehittäjät voivat keskittyvä heidän sovellusten logiikkaan ja riippuvuuksiin, kun taas IT-operatiiviset ryhmät voivat keskittyä käyttöönottoon ja hallintaan vaivautumatta sovelluksen yksityiskohtiin.(Google Cloud – Containers.)

Säiliöitä tai kuvaavammin sovelluskontteja verrataan usein virtuaalikoneisiin, jotka ovat isäntäkäyttöjärjestelmän päällä toimivia vieraskäyttöjärjestelmiä, joilla on virtuaalisoitu pääsy taustalla olevaan laitteistoon. Kuten virtuaalikoneet, sovelluskontit mahdollistavat sovelluksen pakkauksen yhdessä sen koodikirjastojen ja muiden riippuvuuksien kanssa tarjoten eristetyn ympäristön ohjelmistopalvelun suorittamiselle. Näiden yhtenäisyydet loppuvat tähän, mutta sovelluskontit tarjoavat paljon kevyemmän ympäristön kehittäjille, millä on lukemattomia etuja.




Kuva 7. Virtuaalikoneen ja säiliöinnin eroavaisuudet (Google Cloud – Containers).

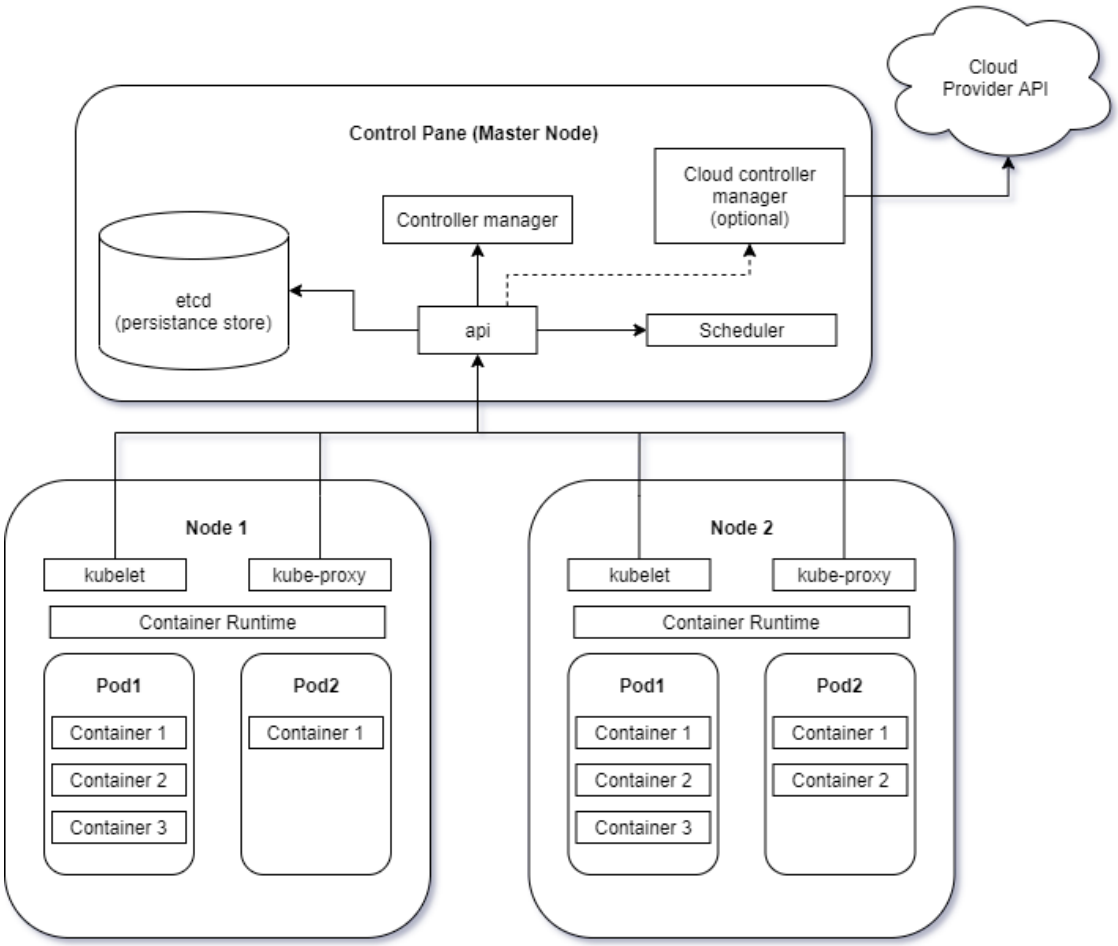
Kuvassa 7 kuvattu virtuaalikoneiden lähestymistavassa virtualisointi on toteutettu oman ohjelmistokerroksen (Hypervisor) kautta, joka suorittaa jokaista virtuaalikonetta. Sen sijaan, että laitteistopino virtualisoitaisiin oman ohjelmistokerroksen kautta, säiliöinnin lähestymistavassa virtualisointi tapahtuu käyttöjärjestelmätasolla niin, että useita sovelluskontteja suoritetaan suoraan käyttöjärjestelmän ytimen päällä. Koska kontit jakavat käyttöjärjestelmän ytimen, ovat ne paljon kevyempiä ja käynnistyvät paljon nopeammin sekä käyttävät murto-osan muistista verrattuna koko käyttöjärjestelmän käynnistämiseen.(Google Cloud – Containers.)

Saatavilla on monia säiliömuotoja, joista suosituin on avoimen lähdekoodin säiliömuoto Docker (Carter 2018).

### 3.2.2 Kubernetes

Kuten voimme aikaisemmasta luvusta päätellä, säiliöinti on hyvä tapa pakata ja käyttöönottaa sovelluksia, mutta lopullisessa ympäristössä on hallittava sovelluskontteja ja varmistettava, ettei seisonta-aikoja ole. Esimerkiksi jos sovellus kaatuu säiliössä, jonkun pitää uudelleen käynnistää säiliö. Tämän kaltainen käyttäytyminen olisi hyvä mahdollistaa järjestelmän hoidettavaksi. (What is kubernetes 2021.)

Kubernetes-hallintaohjelmiston ideana on mahdollistaa automatisoitu Docker-konttien hallinta, joka parantaisi palveluiden luotettavuutta ja vähentäisi päivittäiseen toimintaan liittyvää aikaa ja resursseja. Kubernetes-ohjelmiston keskeisimpiin tehtäviin kuuluu konttisovellusten käyttöönotto ja ylläpito, sovelluskonttien päivittäminen uuteen versioon, resurssien skaalautuminen ylös ja alas tarpeiden mukaan sekä sovellusten monitorointi. Nämä automatisoidut tehtävät helpottavat ja nopeuttavat uusien toteutuksien julkaisemista sekä parantavat palvelun luotettavuutta suorittamalla jatkuvasti säiliöiden kunnon tarkistamista ja uudelleenkäynnistämällä jumissa olevat säiliöt. Kubernetes-hallintaohjelmisto hoitaa myös klusterin resurssinhallinnan, verkkoliikenteen ja tilanhallinnan, mikä mahdollistaa kehittäjien keskittymään sovelluksiin eikä huolehtimaan taustalla olevasta ympäristöstä. (What is kubernetes 2021.)




Kuva 8. Kubernetes-klusterin rakenne.

Kuvasta 8 nähdään, miten Kubernetes-klusteri koostuu joukosta Node-palvelimiksi kutsuttuja työntekijäkoneita (worker machines), jotka suorittavat säiliöityjä sovelluksia. Jokaisella klusterilla on vähintään yksi Node-palvelin (Kubernetes Components 2021). Node-palvelin isännöi Kubernetes-Podeja, jotka ovat käytännössä Docker-konttikokoelmia, jotka jakavat saman levytilan, verkkoresurssit ja säiliöiden suorittamisen määritykset (Kubernetes Pods 2021).

Kubernetes-Podit, tai kuvaavammin kapselit, eivät ole pysyviä resursseja vaan niitä luodaan ja tuhotaan dynaamisesti vastaamaan klusterin tilaa. Jokainen kapseli saa oman IP-osoitteen, mutta jossain toisessa ajan hetkessä samaa sovellusta voi ylläpitää eri kapseli. Jotta tiedetään, missä osoitteessa tietty toiminnallisuus tietyllä ajan hetkellä on klusterissa, tarvitaan klusteriin oma palvelu näiden hallitsemiseen. Kubernetesissa palvelu on abstraktio, joka määrittelee loogisen joukon kapseleita ja käytännön, jolla niitä voidaan käyttää. Joskus edellä mainittua mallia kutsutaan myös mikropalveluksi. Jokaiselle klusterissa määritetylle palvelulle, mukaan lukien itse DNS-palvelimen, määritetään DNS-tietueet, joita voidaan käyttää klusterin IP-osoitteiden sijaan. (Kubernetes Service 2021.)

Jotta Kubernetes tietää, miten kapseleita käynnistetään, tarvitaan niille asennusmääritykset (deployments). Asennusmäärityksellä tässä kontekstissa tarkoitetaan kapselin halutun tilan määrittämistä. Asennusmääritykseen merkitään kaikki asennukseen tarvittavat tiedot, kuten kapselin sisältämien konttien levykuvat ja konttien elävyysanturit, joista kubelet-agentti-niminen palvelu seuraa sen kuntoa ja tarvittaessa käynnistää sovelluskontin uudelleen. (Configure Liveness, Readiness and Startup Probes 2021.)

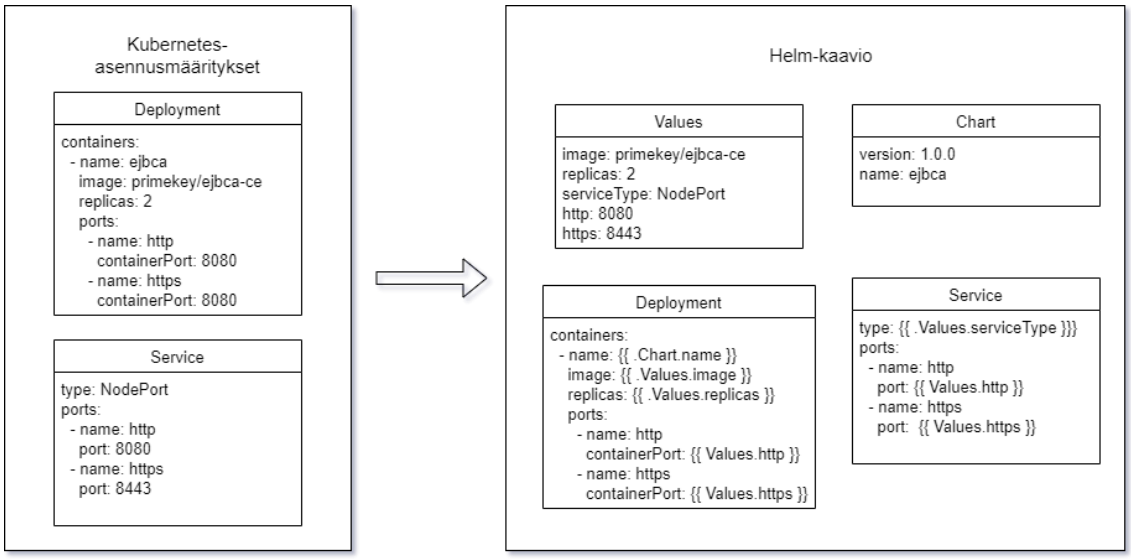
Asennusmääritykset kirjataan YAML (YAML Ain’t Markup Language) -formaattiin, joka on ihmisystävällinen merkintäkieli. Kuten kuvasta 9 nähdään, YAML on

JSON-formaatista jatkettu merkintäkieli. JSON-tiedostot ovat siis valideja YAML 1.2-formaatin tiedostoja. (Oren ym. 2009.)

| XML | JSON | YAML |
|---|---|---|
| <Servers><br><Server><br><name>Server1</name><br><owner>John</owner><br><created>123456</created><br><status>active</status><br></Server><br></Servers> | {<br>Servers: [<br>{<br>name: Server1,<br>owner: John,<br>created: 123456,<br>status: active<br>}<br>]<br>} | Servers:<br>- name: Server1<br>owner: John<br>created: 123456<br>status: active |

Kuva 9. JSON- ja YAML-merkintäkielien erot (Deshpande 2019).

Koska Kubernetes-asennusmäärityksistä voi tulla helposti laajoja, olisi hyvä jakaa näitä uudelleenkäytettäviksi paketeiksi, mitkä voidaan helposti päivittää, asentaa ja poistaa. Tähän ratkaisuna on Helm-pakettihallintajärjestelmä, jolla voidaan paketoida asennusmääritys tar (tape archive) -tiedostoon. Helm-pakettihallintajärjestelmän avulla asennusmäärityksistä voidaan tehdä monipuolisia kaaviota, joiden käyttökohtaisia asetuksia voidaan määrittää erillisessä tiedostossa. Tämä mahdollistaa saman kaavion käyttämisen useissa eri ympäristöissä. Helm-pakettihallintajärjestelmä on siis suuri apu Kubernetesin hallinnassa, mutta suurin etu siitä on CI/CD-putkistojen virtaviivaistamisessa. (Using Helm 2021.)




Kuva 10. Yksinkertaistettu kuvaus Helm-pakettijärjestelmän hyödystä.

Kuvasta 10 nähdään, kuinka Helm-pakettijärjestelmässä asennusmääritysten arvot siirretään omaan ”Values”-tiedostoon. Asennusmääritykset ovat näin helposti löydettävissä yhdestä tiedostosta ja muutettavissa kohdeympäristön tarpeisiin. Lisäksi koska Helm-pakettijärjestelmä on kirjoitettu Go-ohjelmointikielellä, on mahdollista käyttää sen koodikirjastoja YAML-tiedostoissa monipuolisten ja uudelleenkäytettävien mallipohjien luomiseen.

## 3.3 Palvelun rakenteen suunnittelu

### 3.3.1 Ohjelmistojen valinta

Suurimpana tarpeena ohjelmistojen valinnassa oli etsiä avoimen lähdekoodin toteutus PKI-ohjelmistosta, joka tukisi CMP- ja LDAP-protokollia. Ainoaksi soveltuvaksi ohjelmistoksi osoittautui ruotsalaisen PrimeKey Solutionsin ylläpitämä Javalla kirjoitettu EJBCA, jonka koodipohjasta PrimeKey omistaa suurimman osan. EJBCA on lisensoitu ja käytettävissä LGPL (GNU Lesser General Public Lisence) ehtojen mukaisesti. EJBCA-ohjelmiston kattavien mahdollisuuk-

sien ja vaadittujen ominaisuuksien lisäksi PrimeKey toimittaa ohjelmistoa valmiiksi Docker-kontissa, eli EJBCA-ohjelmistopäivitykset ovat siis vaivatta käyttöönotettavissa Kubernetes-ympäristössä eivätkä vaadi mitään ylläpitoa.

PKI-ratkaisun voi toteuttaa ja suunnitella monella tapaa, yksinkertaisista ja edullisista, erittäin monimutkaisiin ja kalliisiin. EJBCA mahdollistaa käytännössä minkä tahansa tyyppisen PKI-arkkitehtuurin toteuttamisen. Yksi EJBCA-asennus voi toimia samanaikaisesti molemmissa CA (Certificate Authority) ja RA (Registration Authority) -asemissa. Tämä on hyvin tehokas ja helposti hallittava ratkaisu, joka on hyvin soveltuva testitarpeisiin. EJBCA myös mahdollistaa useamman CA-yksikön sijaitsemisen samassa asennuksessa, mikä mahdollistaa erityyppisten luottamusketjujen rakentamisen testitapauksiin.

Koska EJBCA-ohjelma ei itsessään toteuta hakemistopalvelua käyttäen LDAP-protokollaa, joudutaan etsimään LDAP-implementaatio, joka käynnistetään EJBCA-ohjelmiston rinnalla. Implementaation olisi hyvä olla avoimen lähdekoodin lisäksi nopeasti säiliössä käynnistettävä kevyt ohjelmisto. Näistä löytyi kaksi potentiaalista implementaatiota OpenLDAP ja OpenDJ, joista Open Indentity Platform Community ylläpitämä OpenDJ osoittautui soveltuvan paremmin projektin tarpeisiin. OpenDJ on jokseenkin uudempi LDAP-implementaatio, missä on panostettu käyttöönoton helppouteen, yksinkertaisuuteen ja korkeaan suorituskykyyn.

EJBCA-ohjelmisto tarvitsee relaatiotietokannan tietojen tallentamiseen, ja vaikka ohjelmiston mukana tulee H2-muistitietokantajärjestelmä, joka periaatteessa olisi soveltuva tämän projektin käyttötarpeisiin sen nopean käyttöönoton ja tallennustavan takia, on sitä työläämpi hallita ulkopuolelta. Tästä syystä tietokannaksi tuli valittua JDBC-ajuria tukeva ja PrimeKeyn suosittelema MariaDB MySQL-relaatiotietokanta.

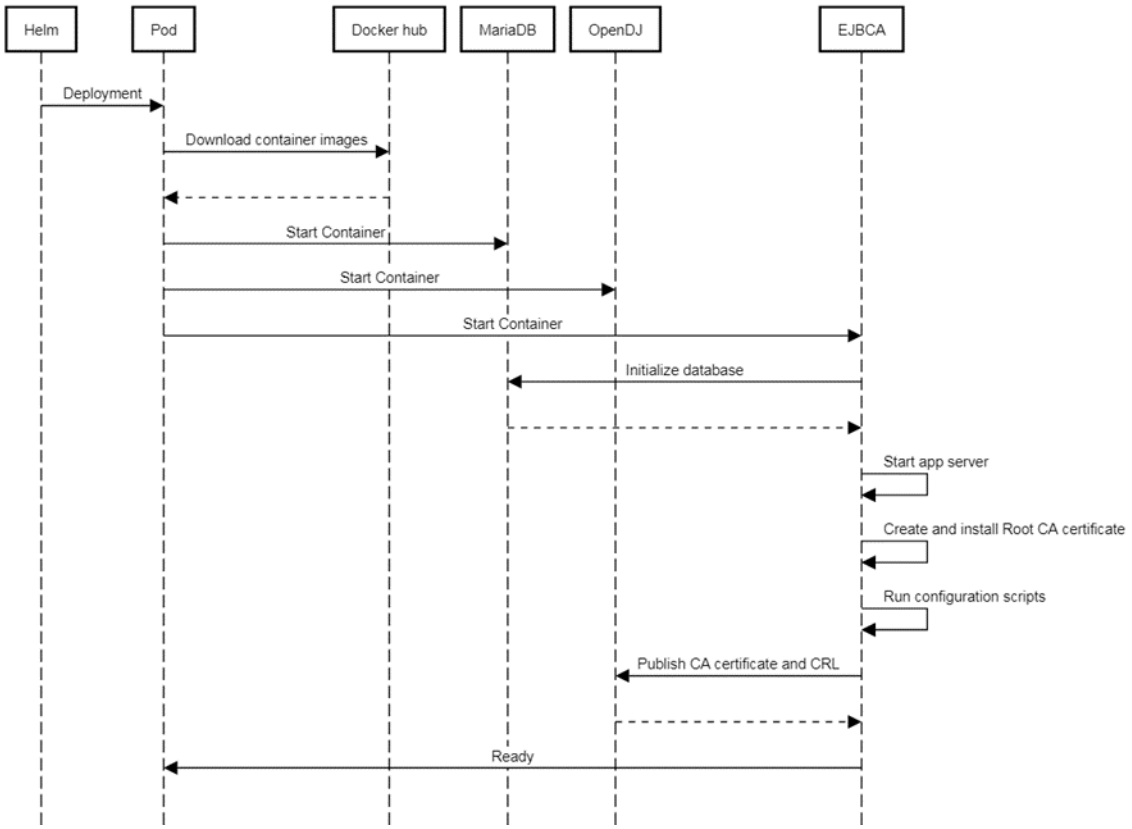
Näiden konttien suorittamiseen lokaalisti tarvitaan jokin Kubernetes-jakelu. Insinöörityössä käytettiin Rancher Labsin kehittämää K3s:aa, joka on täysin yh-

teensopiva Kubernetes-jakelu. Muita suosittuja kehitystarkoitukseen suunniteltuja Kuberenetes-jakeluja ovat esimerkiksi Minikube ja MicroK8s. K3s-jakelun etu näihin on, että se voidaan suorittaa missä tahansa Linux-järjestelmässä ilman ylimääräisiä riippuvuuksia tai työkaluja, kuten virtualisointiohjelmistoa, mikä tekee tästä kevyen ja soveltuvan testaustarkoitukseen.

### 3.3.2 Ohjelmistojen asennusmääritys

Jotta nämä sovelluskontit saadaan suoriutumaan Kubernetes-klusterissa halutusti, pitää määrittää palvelulle asennusmääritys. Tässä tarpeena on ymmärtää näiden väliset kommunikaatiot ja ohjelmistojen käynnistystavat.

EJBCA-ohjelmisto voidaan suorittaan millä tahansa JEE5-kehitysalustaa tukevalla sovelluspalvelimella, mutta tämä projektin tapauksessa PrimeKeyn toimittama sovelluskontti käyttää Wildfly 12 -sovelluspalvelinta. Ennen sovelluspalvelimen käynnistämistä tarvitsee olla tietokanta alustettuna, minkä takia MariaDB tulisi käynnistää ensimmäisenä. EJBCA-toiminta on myös riippuvainen LDAP-palvelimesta. Siksi OpenDJ olisi hyvä myös käynnistää tässä vaiheessa. Kuvassa 11 on kuvattu suunniteltua säiliön käynnistyksen kulkua.




Kuva 11. Suunniteltu Kubernetes-kapselin (Pod) käynnistys.

Koska varmentajapalvelu tulisi olla valmiiksi määritelty haluttuun käyttötarpeeseen, tulisi EJBCA-ohjelmiston konfiguraatiot olla tehtynä ennen käyttöönvalmiutta. EJBCA-ohjelmisto mahdollistaa konfiguraatioiden tuonnin sen komentotulkin kautta, millä tämä voitaisiin toteuttaa. Toinen vaihtoehto olisi dumpata konfiguraatiot suoraan tietokantaan, mutta haittana tässä vaihtoehdossa olisi tietojen validoimattomuus, joka mahdollisesti aiheuttaa sovelluksen kaatumista. Kubernetes mahdollistaa konfiguraatiotiedostojen asentamisen säiliöiden sisään, jolloin voitaisiin esimerkiksi asettaa säiliöön Shell-komentosarja, joka suoritetaan EJBCA-kontin käynnistyessä ja joka suorittaisi kaikki tarvittavat konfiguraatiot CA-palveluun.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: "configuration-script"
data:
{{ tpl (.Files.Glob "./script.sh").AsConfig . | indent 2 }}
```

Esimerkki 1. Kubernetes ConfigMap-objekti.

Konfiguraatiotiedostojen asentaminen säilöön voidaan toteuttaa luomalla edellä kuvattu ConfigMap-objekti. ConfigMap on Kubernetes API-objekti, joka mahdollistaa ei-luottamuksellisen tiedon tallentamisen avainarvopareihin (Kubernetes – ConfigMaps). Käytännössä esimerkissä oleva Shell-komentosarjan (script.sh) sisältö luetaan ja tallennetaan ”data”-avaimeen, mitä Kubernetes-kapseli pystyy käyttämään tiedoston luomiseen säiliön sisällä. Heikkoutena ConfigMap-objektissa on sen rajoitteisuus. ConfigMap-objektia ei ole suunniteltu sisältämään suurta määrää dataa, vaan on rajoituttu yhteen mebitavuun (MiB, 1024^2 tavua) (Kubernetes – ConfigMaps). Nykyisiin tarpeisiin ConfigMap-objekti on riittävä, mutta mikäli tulevaisuudessa tarvitaan liittää kymmeniä eri konfiguraatiotiedostoja, olisi hyvä miettiä myös vaihtoehtoisia ratkaisuja. Konfiguraatiot voitaisiin esimerkiksi kopioida isäntäkoneelta säiliöön käyttäen Kubernetes-ohjelmointirajapintaa kapselin käynnistyksen jälkeen, mutta ennen testien suorittamista.

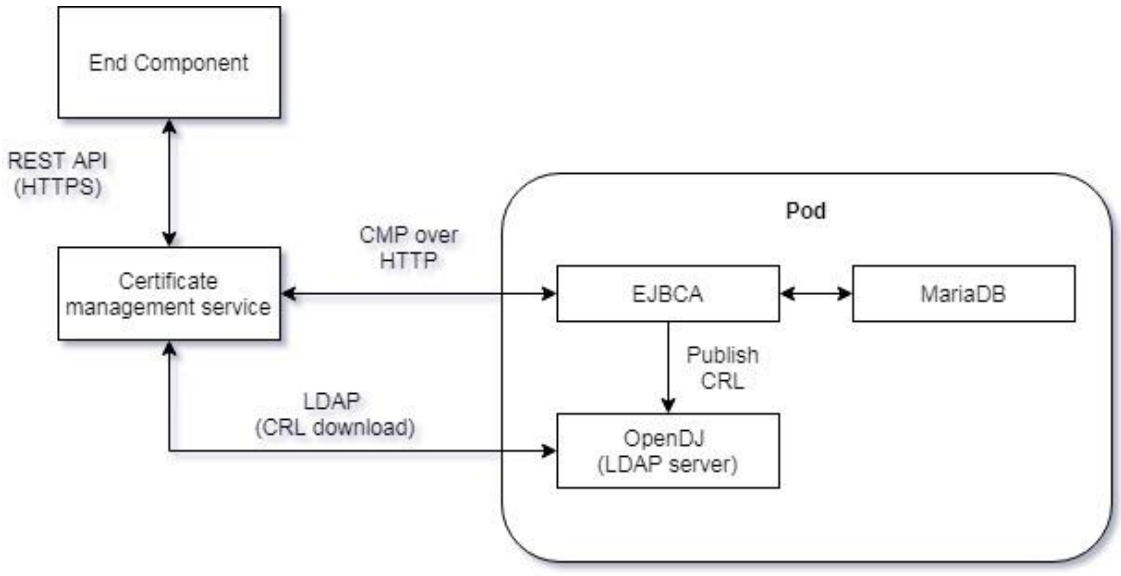
Jotta Kubernetes-kapselissa oleva kubelet-palvelu tietää, mistä tarkistaa säiliön kunnon, tarvitsee näille määrittää anturit. Anturilla tässä kontekstissa tarkoitetaan tapaa, jolla voidaan varmistaa sovelluskontin vastaavan kutsuihin, eikä ole jäänyt jumiin. Esimerkiksi EJBCA-ohjelmistossa on HTTP-palvelu, josta voidaan monitoroida ohjelmiston kuntoa. Tämän palvelun osoite ja tarkastuksen intervallit voidaan määrittää asennusmäärityksissä, jolloin kubelet-palvelu osaa uudelleen käynnistää sovelluskontin, jos se ei vastaa pyyntöihin määritellyssä ajassa. Konteille voidaan määrittää kolme eri tyyppistä anturia:

- Elävyys (liveness), josta kubelet-palvelu tietää, milloin sovellus on jäänyt jumiin ja tarvitsee käynnistää uudelleen.

- Valmius (readiness), josta kubelet-palvelu tietää, milloin sovellus on valmiina hyväksymään ulkoista liikennettä.

- Käynnistys (startup), josta kubelet-palvelu tietää sovelluksen käynnistyneen.

Koska sovelluskontin elävyyttä ja valmiutta ei välttämättä heti haluta varmistaa sen käynnistyessä, voidaan konteille määrittää käynnistysanturit. Jos käynnistysanturi on määritelty, kubelet-palvelu ei tarkista kontin elävyyttä tai valmiutta, vaan suorittaa käynnistysanturissa määriteltyä kyselyä, kunnes saa positiivisen vastauksen. Käytännössä käynnistysanturille määritellään, montako kertaa kysely voi epäonnistua käynnistyksen aikana. Jos määritelty epäonnistumisten määrä ylittyy, voidaan todeta, että käynnistys on epäonnistunut ja kubelet-palvelu voi käynnistää sovelluskontin uudestaan.




Kuva 12. Suunnitellun kapselin (Pod) rakenne.

Kuvassa 12 on esitetty rakennetun kapselin (Pod) rakennetta ja kytköksiä muihin komponentteihin. Jotta EJBCA tietää, mistä löytää muut kapseliin asennetut

palvelut, nämä pitää määrittää asennusmäärityksissä. Tarvittavat tiedot MariaDB- ja OpenDJ-palveluista, kuten osoitteet ja käyttäjätiedot, voitiin helposti määrittää valmiille EJBCA Docker -levykuvalle ympäristömuuttujina.

# 4 CA-palvelimen konfigurointi

Jotta saadaan asennuksen aikana konfiguraatiot määritettyä sertifikaattihallintapalvelun testitapausten suorittamista varten, tarvitsee tutkia EJBCA-ohjelmiston konfiguraatioita ja mahdollisuuksia. Konfigurointiin liittyi käytännössä kolme eri osa-aluetta:

1. Varmenneprofiilit, joissa määritellään luotavien varmenteiden rakenne.

2. CMP-protokollan määritykset, kuten mitä CMP-vastausviesteihin sisällytetään ja mitä suojausmenetelmää missäkin kutsussa käytetään.

3. Julkaisijan, eli julkisten hylkäyslistojen ja varmenteiden arkistoa ylläpitävän palvelun määritykset.

Ohjelmiston mahdollisuuksien ja konfiguraatioiden testausta nopeutti huomattavasti edellisessä luvussa suunniteltu asennusmääritys, minkä avulla palvelu voitiin käynnistää aina tarvittaessa uudestaan tyhjillä asetuksilla parissa minuutissa.

## 4.1 Varmenneprofiilit

Varmenteiden sisältö riippuu monesta eri tekijästä. Esimerkiksi varmentajilla, välivarmentajilla ja loppukäyttäjällä on erityyppiset varmenteet. Lisäksi X.509-standardin kolmas versio myös mahdollistaa useita eri laajennuksia, joita voidaan varmenteisiin asettaa. Varmenteen tilaaja voi vaatia varmenteeseensa jotain tiettyjä X.509-standardin laajennuksia, joita muut eivät välttämättä vaadi. Varmenteisiin tyypillisesti luodaan profiileja eri käyttötarkoituksiin. Näissä määritellään asioita kuten sitä, mihin myönnettävää varmennetta voidaan käyttää,

kauanko se on voimassa, mitä allekirjoitusalgoritmia käytetään ja mitä laajennuksia varmenteessa on käytössä.

Nykyisiin testitarpeisiin luotiin useamman varmentajan pituinen luottamisketju. Välivarmentajien varmenteille luotiin omat profiilit, missä määriteltiin esimerkiksi, kuka allekirjoittaa varmenteen ja varmenteen rajoitukset. Loppukäyttäjälle määriteltiin myös yleinen profiili, jota kaikki luodut varmentajat voivat käyttää uusien varmenteiden luomiseen.

Varmentajia EJBCA-ohjelmistossa pystytään luomaan komentorivillä yhdellä komennolla:

```
ejbca.sh ca init \
    --caname "SubCA_1" \
    --dn "\CN=SubCA_1\O=Example\C=FI" \
    --tokenType "soft" \
    --tokenPass "null" \
    --keytype "RSA" \
    --keyspec "1024" \
    -v "365" \
    -s "SHA256WithRSA" \
    --signedby ${ROOTCA_ID} \
    -certprofile "SUBCA_1_PROFILE"
```

Esimerkki 2. CA-instanssin luonti EJBCA-komentotulkilla.

Esimerkkikomennossa varmentajalle määritellään ohjelmistoa varten nimi (caname) ja sen tunnistettava nimi (DN, Distinguished Name). Komennossa myös tulee määrittää, minkä tyyppisessä tunnuksessa salausavaimet säilytetään (tokenType). EJBCA-ohjelmistossa on säilyttämiselle kaksi vaihtoehtoa käyttäen HSM (Hardware Secure Module) -moduulia, joka on arvokkaiden salausavaimien luomiseen, tallentamiseen ja suojaamiseen suunniteltu fyysinen laite, tai ohjelmiston avainvarastoa (esimerkissä märitetty arvolla ”soft”), joka on salattu tiedosto tietokannassa (Jochen 2019). Tässä insinöörityössä HSM-moduulia ei ole käytössä, joten käytettiin ohjelmiston avainvarastoa tallettamiseen. Tunnukselle tulee myös määrittää salasana (tokenPass), joka voidaan asettaa

käyttämään järjestelmän oletussalasanaa määrittämällä ”null”, tai määrittää arvoksi ”prompt”, milloin voidaan tämä itse asettaa komentotulkista. Muita pakollisia määrityksiä varmentajan luomisessa ovat allekirjoitukseen käytettävä salausavaimen algoritmi (keytype) ja sen pituus (keyspec), varmenteen voimassaoloaika päivissä (-v, validity) ja allekirjoituksen algoritmi (esimerkissä määritelty -s argumentilla). Vapaaehtoisia määrityksiä on useita, mutta tässä tapauksessa ei ollut tarvetta käyttää muita kuin allekirjoittajan (signedby) ja muokatun varmentajaprofiilin (certprofile) määrityksiä.

Tulevia testitapauksia varten piti myös tehdä tapa, miten profiileja voidaan helposti luoda ja lisätä Helm-kaavion asetusmäärityksiin. Tätä varten rakennettiin Shell-komentosarja, jolla voitiin helposti tuoda profiilit sovelluskontista suoraan kaavion hakemistoon isäntäkoneelle. Tiedostojen tuomisen mahdollisti Kubernetes-komentorivityökalu kubectl, jolla pystytään suorittamaan komentoja sovelluskonttien sisällä ja kopiomaan tiedostoja kontista isäntäkoneelle.

## 4.2 Varmenteiden hallintaprotokolla

Kuten luvussa 2 mainittiin, varmenteilla on voimassaoloaika, joka jossakin vaiheessa päättyy. Näiden digitaalisten varmenteiden elinkaaret olisi hyvä pystyä automatisoimaan. PKI-entiteettien väliseen X.509-standardin digitaalisten varmenteiden hankkimiseen ja hallintaan on suunniteltu hallintaprotokollia, joista käytetyin on RFC4210:ssä kuvattu CMP (Certificate Management Protocol) -protokolla. CMP-protokolla on yksi kahdesta varmenteiden hallintaprotokollasta, jotka käyttävät RFC4211-dokumentissa kuvattua CRMF (Certificate Request Message Format) -viestimuotoa (EJBCA – Certificate Management Protocol). Toinen CRMF-viestimuotoa käyttävä hallintaprotokolla on RFC5272-dokumentissa kuvattu CMC (Certificate Management over Cryptographic message syntax).

CMP-protokolla ei siis ole ainoa varmenteiden hallintaan käytetty protokolla, mutta 3GPP (Third Generation Partnership Project), joka on mobiililaajakaistastandardi, on valinnut CMP-protokollan varmenteiden hallintaan. 3GPP TS

33.310-standardi [2021: 41-44] määrittää CMPv2-profiilin, joka on RFC4210-dokumentin määrityksistä jatkettu täsmällisempi versio.

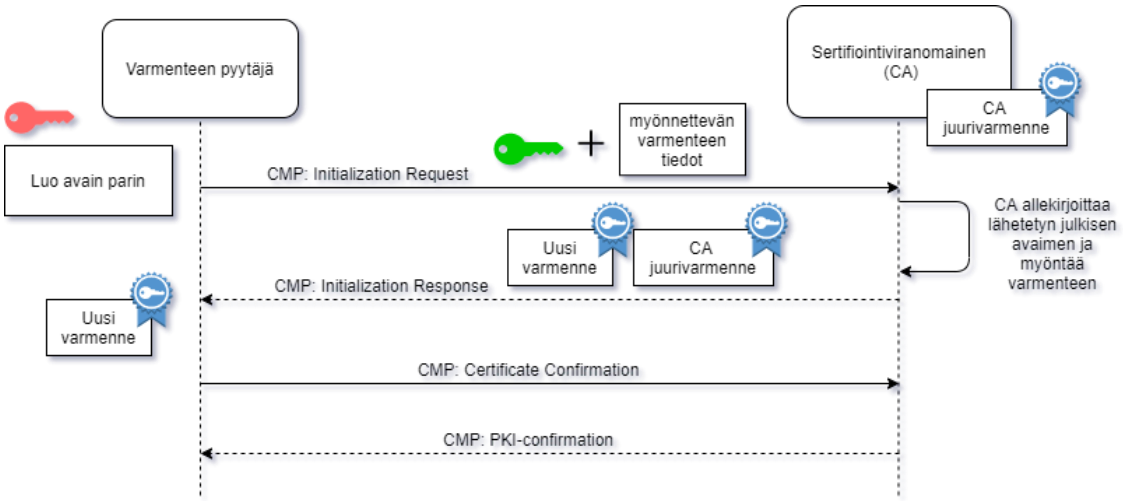
CMP-protokolla koostuu pääasiassa seuraavista operaatioista:

- **Initialization Request** entiteetin rekisteröitymiseen
- **Certificication Request** lisävarmenteiden hankkimiseen olemassa olevalle entiteetille
- **Key Update Request** varmenteen päivittämiseen
- **Key Recovery Request** avainten palautukseen
- **Revocation Request** varmenteen mitätöimiseen.

Koska insinöörityöprojektin kohdepalvelu toteuttaa 3GPP-standardin [2021: 41-44] CMPv2-profiilia, joka sisältää vain varmenteen rekisteröitymispyynnön (Initialization Request) ja päivityspyynnön (Key Update Request), ei tässä insinöörityössä keskitytä muihin RFC4210-dokumentissa kuvattuihin pyyntöihin.

### 4.2.1 Varmenteiden rekisteröitymis- ja päivitysprosessi

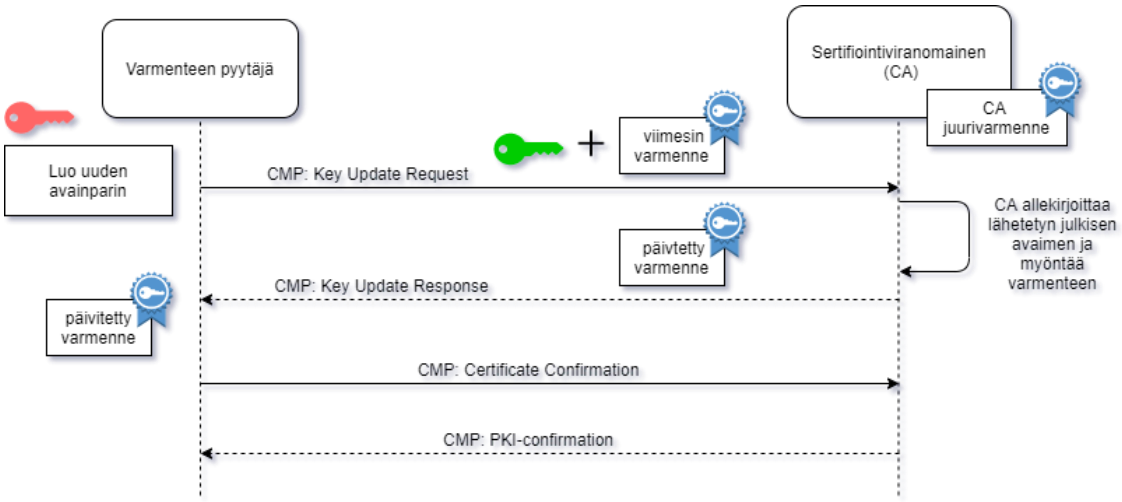
Varmenteiden rekisteröitymisellä tarkoitetaan tilannetta, jossa jokin entiteetti luo oman julkisen avainparin ja haluaa jakaa avainparin julkista avainta varmenteen muodossa. Tämänkaltaisessa tilanteessa sertifiointiviranomaisella ei ole vielä omissa tiedoissa rekisteröintiä pyytävän entiteetin tietoja ja rekisteröitävällä entiteetillä ei ole välttämättä asennettuna varmentajan varmennetta. Rekisteröitymisprosessia varten on määritelty sen kulku, joka on kuvattu RFC4210-dokumentissa [2005: 38].




Kuva 13. CMP-protokollan rekisteröintipyynnön kulku.

Kuvassa 13 on kuvattu varmenteen rekisteröintioperaation kulkua. Varmenteen pyytäjä, joka tyypillisesti on IoT(Internet of Things) -laite, luo aluksi uuden epäsymmetrisen avainparin ja lähettää CMP-protokollan mukaisen rekisteröitymispyynnön (Initialization Request). Pyyntöviestiin sisällytetään luodun avainparin julkinen salausavain (kuvattu vihreällä) ja varmenteen luomiseen tarvittavat tiedot, kuten tunnistettava nimi, jolle varmenne myönnetään. Varmentaja allekirjoittaa vastaanotetun julkisen avaimen ja myöntää varmenteen annettujen tietojen perusteella. Myöntämisen jälkeen varmentaja lähettää CMP-protokollan mukaisen rekisteröitymispyynnön vastauksen (Initialization Response), joka sisältää laitteelle myönnetyn varmenteen ja juurivarmenteen. Kuvassa oleva sertifiointiviranomainen ei välttämättä ole juurivarmentaja, jonka varmenne paluuviestissä lähetetään, vaan voi olla keskiasteen varmentaja. Tällöin allekirjoitettava sertifiointiviranomainen voi sisällyttää oman varmenteen lisäksi muita varmenteita, joita laite saattaa tarvita luottamusketjun varmistamiseksi. Kun laite on saanut vastauksessa uuden varmenteensa, tarkistetaan, onko vastaanotettu varmenne hyväksyttävä. Eli käytännössä varmenteen saanut laite ilmoittaa sertifiointiviranomaiselle, että varmenne on vastaanotettu ja on hyväksyttävä. Viimeinen viesti on varmentajan kuittaus tästä ilmoituksesta. (Adams ym. 2005: 37-42.)

Koska rekisteröidylle varmenteelle määritetään voimassaoloaika, tarvitsee sitä säännöllisesti päivittää. Ennen voimassaoloajan päättymistä voidaan suorittaa päivityspyyntö sertifiointiviranomaiselle, joka myöntää päivitetyn version varmenteesta.




Kuva 14. CMP-protokollan päivityspyynnön kulku.

Kuten kuvasta 14 nähdään, päivitysprosessin kulku on hyvin samanlainen kuin varmenteen pyyntöprosessissa. Poikkeuksena päivitysprosessissa lähettäjä sisällyttää viimeisimmän sertifiointiviranomaisen allekirjoittaman varmenteen ensimmäiseen viestiin. Vaikka kuvassa ei vastausviestiin ole sisällytetty juurivarmennetta, voi sertifiointiviranomainen halutessa sen sisällyttää, mutta esimerkiksi 3GPP TS 33.310-standardissa [2021: 41-44] määritelty CMPv2-profiili ei tätä sisällytä. Juurivarmenne pitäisi olla jo asennettuna varmennepyynnön jälkeen, ja juurivarmenteen lisääminen vastausviestiin olisi turhaa.

Vaikka kaikki rekisteröitymispyynnön ja päivityspyynnön tieto on julkista, tarvitaan silti salausmenetelmiä, jotta voidaan varmistaa myöntävän varmenteen sertifiointiviranomaisen aitous. CMP-protokollassa varmistamiselle on kaksi eri tapaa. Ensimmäinen tapa on käyttää myöntävän sertifiointiviranomaisen varmennetta ja sen sisältävää julkista avainta vastausviestien allekirjoitusten todentamiseen. (Adams ym. 2005: 28.)

Tapauksissa, joissa laitteella ei vielä ole myöntävän sertifiointiviranomaisen varmennetta asennettuna, voidaan käyttää valmiiksi jaettua salasanaa. Tämänkaltaista viestien suojausmenetelmää kutsutaan nimellä PSK (Pre-Shared Key), PBE (Password Based Encryption) tai PBM (Password-Based MAC). (Java Cryptography Architecture Standard Algorithm Name Documentation – MAC Algorithms.)

PBM-suojausmenetelmä on toteutettu käyttäen jotain standardoitua salasanapohjaista hajautusfunktiota ja menetelmiä suojauksen luomiseksi. Esimerkkejä käytetyistä hakautusfunktioista ovat RFC2898-dokumentissa esitetty PKCS#5 (Public Key Crypthography Standard) ja RFC7292-dokumentissa esitetty PCKS#12. Kuten seuraavassa esimerkistä nähdään, paluuviestit validoidaan aina sertifiointiviranomaisen oikeellisuuden varmistamiseksi.

```
cmp_main:apps/cmp.c:2727:CMP info: using section(s) 'ejbca,ir' of OpenSSL con-
figuration file 'openssl.cnf'
setup_client_ctx:apps/cmp.c:2044:CMP info: will contact
http://127.0.0.1:30080/ejbca/publicweb/cmp/cmpRA/
CMP DEBUG: Starting new transaction with
ID=F9:8F:71:BA:22:3A:59:02:46:3A:9E:CF:52:83:C8:56
CMP info: sending IR
CMP DEBUG: connecting to CMP server 127.0.0.1
CMP DEBUG: disconnected from CMP server
CMP info: received IP
CMP DEBUG: validating CMP message
CMP DEBUG: sucessfully validated PBM-based CMP message protection
CMP DEBUG: trying to build chain for newly enrolled cert
CMP DEBUG: success building approximate chain for newly enrolled cert
CMP info: sending CERTCONF
CMP DEBUG: connecting to CMP server 127.0.0.1
CMP DEBUG: disconnected from CMP server
CMP info: received PKICONF
CMP DEBUG: validating CMP message
CMP DEBUG: sucessfully validated PBM-based CMP message protection
save_free_certs:apps/cmp.c:2093:CMP info: received 1 CA certificate(s)
save_free_certs:apps/cmp.c:2093:CMP info: received 1 enrolled certificate(s)
```

Esimerkki 3. Konsoliote CMP-protokollan rekisteröitymispyynnöstä OpenSSL CMP-työkalulla.

### 4.2.2 PKI-hallintaviestien rakenne

Koska tarvittaville PKI-hallintaviesteille pitää määritellä rakenne, on nämä kuvattu RFC4210-dokumentissa [2005: 23]. Kaikki PKI-hallintaan käytettävät viestit käyttävät esimerkissä 4 esitettyä rakennetta.

```
PKIMessage ::= SEQUENCE {
    header           PKIHeader,
    body             PKIBody,
    protection   [0] PKIProtection OPTIONAL,
    extraCerts   [1] SEQUENCE SIZE (1..MAX) OF CMPCertificate
                     OPTIONAL
}
```

Esimerkki 4. RFC4210-dokumentissa [2005: 23] kuvattu PKI-hallintaviestin rakenne.

PKI-hallintaviestin otsikko (header) sisältää monille hallintaviesteille yhtenäisiä tietoja, kuten lähettäjän ja vastaanottajan. Runko (body) sisältää hallintaviestin tyyppikohtaisen numeron. Rungon sisältö riippuu hallintaviestin tyypistä, mitkä on numeroitu 0-27. IR- (Initialzation Request) ja KUR (Key Update request) -kutsut käyttävät RFC4211-dokumentissa kuvatun CRMF (Certificate Request Message Format) -formaatin CertReqMsg-rakennetta, mikä nähdään kuvassa 14.


Kuva 15. Wireshark-verkkoprotokolla-analysaattorilla otettu tapahtuma CMP-protokollan IR (Initialization Request) -viestistä.

CertReqMsg-rakenne sisältää varmenteen niin sanotun mallipohjan, mikä koostuu varmennetta hakevasta julkisesta avaimesta ja avainparin haltijan tiedoista, kuten nimestä (common name), organisaatiosta ja kohdemaasta. CertReqMsg-rakenteen sisältää myös PoP (Proof of Posesssion) -kentän, joka sisältää pyynnön tekevän entiteetin allekirjoituksen avainparin haltijan todistamiseksi. Kuvassa 15 PoP-kenttä on nimellä ”popo”. Tämä kenttä on vapaaehtoinen RFC4210-standradissa [2005: 17], mutta vaadittu 3GPP-standardissa [2021: 38].

Jos sertifiointiviranomainen suorittaa IR- ja KUR-pyynnöt hyväksytysti, silloin IP (Initialization Response) ja KUP (Key update Response) sisältävät vastausviestissä myönnetyn varmenteen ja mahdollisen juurivarmenteen. Muissa tapauksissa paluuvistissä on vain kerrottu status. Tyypillisesti virheellisten pyyntöjen vastausviesti ei myöskään ole suojattu.


Kuva 16. Wireshark-verkkoprotokolla-analysaattorilla otettu tapahtuma CMP-protokollan IP (Initialization Response) -viestistä.

Osa PKI-viesteistä pitää suojata kiistämättömyyden todistamiseksi ja tätä varten viestissä on määritelty suojaus (protection). Suojaus on viesteissä vapaaehtoinen, mutta 3GPP-standardissa [2021: 42] tämä on vaadittu. Jos viestin suojaus on käytössä kuten kuvissa 15 ja 16, sisältää suojauskenttä salatun bittijonon, jota käytetään viestien kiistämättömyyden todistamiseksi.

Viimeisenä olevaan esimerkissä 4 esitettyyn extraCerts-kenttään voidaan sisällyttää varmenteita, joista voi olla vastaanottajalle hyötyä. Esimerkiksi sertifiointiviranomainen voi tähän kenttään lisätä varmenteita, joita loppukäyttäjä tarvitsee oman uuden varmenteensa tarkistamiseen. Huomioon tässä kentässä tulee ottaa, että se ei välttämättä sisällä luottamusketjun sertifiointireittiä, vaan vastaanottaja voi joutua lajittelemaan ja valitsemaan, tai jotenkin muuten prosessoi-

maan ylimääräiset varmenteet käyttääkseen niitä. extraCerts-kenttä on vapaaehtoinen ja voidaan myös jättää pois viestistä, kuten kuvan 16 vastausviestissä on tehty.

### 4.2.3 OpenSSL 3

Insinöörityössä tarvittiin CMP-kutsujen testaukseen CMP-asiakkaan simuloimiseen kätevää työkalua, millä CMP-määrityksiä voitiin testata. Insinöörityössä käytettiin kolmatta OpenSSL-versiota, joka on seuraava kehitteillä oleva OpenSSL-julkaisu. OpenSSL on avoimen lähdekoodin monipuolinen työkalupaketti TLS- (Transport Security Layer) ja SSL (Secure Socket Layer) -protokollille. Toisin kuin edeltävä julkaistu versio 1.1.1, kolmas versio sisältää implementaation RFC4210-, RFC4211- ja RFC6712-dokumenteissa määritellystä CMPv2-protokollasta.

Kolmannen OpenSSL-version CMP-implementaatiota voidaan käyttää varmenteiden pyytämiseen, varmenteiden päivittämiseen, varmenteiden mitätöimiseen ja muun tyyppisten CMP-pyyntöjen suorittamiseen. Versiossa on myös mahdollista tehdä valmiit konfiguraatiot tietyn CMP-palvelimen pyyntöjä varten, mikä mahdollistaa sen kätevän käyttämisen testiautomaatioon. Konfiguraatiot voidaan kirjata esimerkin 5 mukaisesti konfiguraatiotiedostoon.

```
[ejbca]
server = 127.0.0.1:30080
recipient = "/CN=RootCA/O=Example/C=FI"

[ir]
cmd = ir
path = ejbca/publicweb/cmp/cmpRA/
ref = 1234
secret = pass:pBZg-HJZd-p5kq-EBfP
subject = "/CN=Test Subject"
newkey = new_key.pem
certout = new_cert.pem
cacertsout = ca_cert.pem
```

Esimerkki 5. OpenSSL CMP -asetusten määrittäminen.

Konfiguraatiot on mahdollista jakaa kappaleisiin, jolloin ne ovat uudelleen käytettävissä ja kutsut rakenneltavissa eri osioista. Esimerkissä 5 ”ejbca”-osioon on määritelty vain osoitteet ja vastaanottajan tunnistettavan nimen, joka tässä tapauksessa on juurivarmentaja. ”ir”-osioon sen sijaan on kirjattu rekisteröintipyyntöön tarvittavat tiedot. Tämä helpottaa huomattavasti, kun ei tarvitse openssl-komennolle antaa kaikkia tietoja argumentteina, vaan voidaan rekisteröitymispyyntö suorittaa seuraavasti kahdella openssl-komennolla:

```
openssl genrsa 2048 > new_key.pem
openssl cmp -section ejbca,ir
```

OpenSSL 3.0.0 -lähdekoodi on julkisesti saatavilla ja voidaan helposti asentaa mille tahansa Linux-pohjaiselle käyttöjärjestelmälle. Versio voidaan tietysti myös halutessa asentaa Windows-järjestelmään, mutta vaatii Perl-ohjelmistokielen, ANSI C -kääntäjän ja tarpeelliset koodikirjastot, jotka ovat tyypillisesti Linux-jakeluissa valmiina.

### 4.2.4 CMP-määritykset

Insinöörityössä oli tarpeena tehdä IR- ja KUR-pyyntöjä varten CMP-määritykset, jotka tukisivat 3GPP-standardin vaatimuksia. Toisin kuin nykyinen E2E-testeissä käytetty CA-ohjelmisto, ei EJBCA-ohjelmisto mahdollista useiden eri suojausmenetelmien hyväksymistä yhdestä CMP-osoitteesta. Toisaalta EJBCA-ohjelmistossa on mahdollista luoda useita CMP-profiileja, joihin voidaan tehdä CMP-kutsuja.

EJBCA-ohjelmiston CMP-profiileissa on kaksi eri toimintatilaa, jotka määrittävät, miten kutsuihin reagoidaan. RA (Registration Authority) -tilassa EJBCA toimii rekisteröintiviranomaisena ja toisessa Client-tilassa EJBCA hyväksyy vain kutsuja entiteeteiltä, jotka ovat jo rekisteröityneet ohjelmistoon. Käytännössä tämä tarkoitti insinöörityöprojektissa sitä, että molemmille IR- ja KUR-pyynnöille jouduttiin tekemään omat CMP-profiilit.

Koska sertifikaattihallintapalvelulla ei välttämättä ole asennettuna käytettävän CMP-palvelun varmennetta, joudutaan viestien aitouden varmistamiseksi IR-kutsuissa asettamaan suojausmenetelmäksi PBM (Password-Based MAC). KUR-kutsun tapauksessa CMP-palvelun loppukäyttäjällä on jo tämän palvelun varmenne hallussaan, joten käytetään siinä todentamismenetelmänä digitaalista allekirjoitusta.

EJBCA-ohjelmiston CMP-profiileissa pystytään myös määrittämään allekirjoittava varmentaja ja varmenneprofiili, joita uusien varmenteiden luomiseen käytetään. Tämä mahdollistaa erityyppisten varmenteiden ja luottamusketjujen luomiseen testitapauksiin. Voitaisiin esimerkiksi tehdä testitapausta varten virheellistä sisältöä sisältäviä varmenneprofiileja ja nämä olisivat helposti luotavissa tekemällä IR-kutsu tämän CMP-profiilin osoitteeseen.

Vaikka CMP-profiileja testattaessa OpenSSL CMP-työkalulla profiilit toimivat asianmukaisesti, niin sertifikaattihallintapalvelulla ei asia ollutkaan näin. Ongelmana oli sertifikaattihallintapalvelun eräs asiakasvaatimus, jossa CMP-osoitteen polun tulisi automaattisesti lisätä asetetun CMP-polun loppuun puuttuva vinoviiva ”/”. EJBCA-ohjelmiston CMP-profiilin HTTP-osoitteen loppuosa on EJBCA-dokumentoinnin mukaan ”/ejbca/publicweb/cmp/ALIAS”. Alias on CMP-profiilin nimi eikä tällöin päättyisi vinoviivaan. Nopean EJBCA-lähdekoodin selauksen jälkeen tuli ongelmalle tehtyä hyvin yksinkertainen kiertotapa.

```
private String getAlias(final String pathInfo) {
    final String alias;
    if (pathInfo!=null && pathInfo.length()>1) {
        alias = pathInfo.substring(1);
    } else {
        alias = DEFAULT_CMP_ALIAS;
    }
    return alias;
}
```

Esimerkki 6. EJBCA-ohjelmiston CmpServlet-rajapinnan lähdekoodista kaapattu ”getAlias”-funktio.

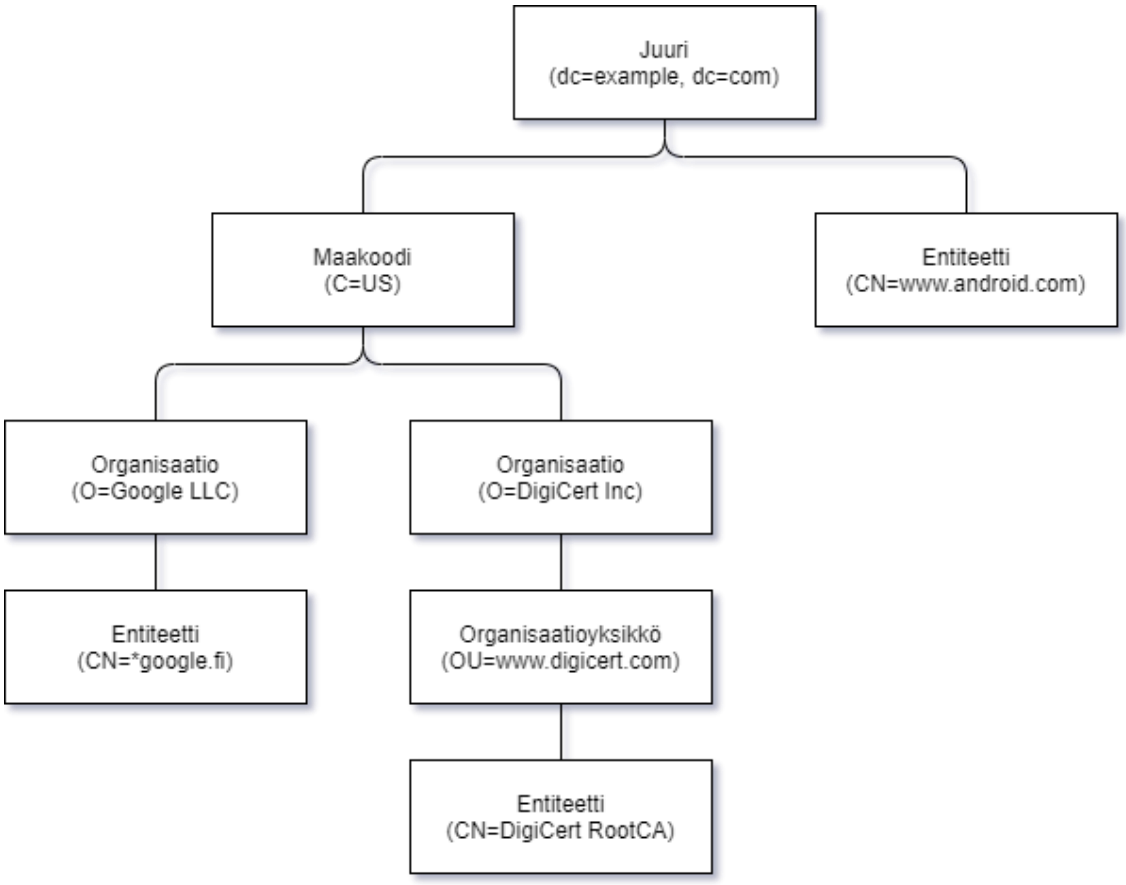
Esitetyn EJBCA-lähdekoodin getAlias funktion pathInfo-argumentti on ”/cmp/” -polun jälkeen erotettu merkkijono. Kuten koodista voidaan päätellä, tämä hakee profiilin millä tahansa merkkijonolla. Havainnon perusteella voidaan CMP-profiilinimien perään vain lisätä tarvittaessa vinoviiva, mikä tällöin kattaisi vaatimuksen.

Koska tarpeena oli, että nämä CMP-määritykset olisivat aina käytössä klusteriympäristöön asennuksen jälkeen, piti kehittää tapa määritysten tuomiseksi EJBCA-ohjelmistoon. EJBCA-komentotulkki tukee CMP-profiiliasetusten tuomisen ”.properties”-tiedostopäätteen tiedostosta. Tämä mahdollisti asetusten kirjaamisen omaan tiedostoon, joka voitiin Helm-kaavion asetusmäärityksissä viedä EJBCA-ohjelmistoon säiliön käynnistyessä.

## 4.3 Julkaisija

Julkisten hylkäyslistojen ja varmenteiden säilöntään tarvitaan arkisto, kuten luvussa 2 puhuttiin. Julkaisija on tässä kontekstissa arkistoa ylläpitävä palvelu. Palvelu voidaan ylläpitää samalla palvelimella kuin varmentaja, mutta tyypillisesti nämä ovat erikseen. Sertifiointiviranomaisen eli varmentajan tehtävänä on tällainen palvelu järjestää varmenteille ja hylkäyslistoille, jotka se on itse allekirjoittanut. (Cooper ym. 2008: 53-54.)

CA-palvelujen suunnittelu -luvussa mainittiin, kuinka sertifikaattihallintapalvelu, joka on insinöörityöprojektin CA-palvelun pääasiallinen käyttäjä, tukee vain LDAP (Lightweight Direct Access Protocol) -protokollaa hylkäyslistojen lataamiseen arkistosta. LDAP-hakemistopalveluprotokolla on TCP/IP-protokollapinossa TCP-protokollan yläpuolella ja tarjoaa mekanismin yhteyden muodostamisen Internet-hakemistoihin ja niiden muokkaamiseen (Sermersheim ym. 2006: 42).




Kuva 17. LDAP-hakemiston rakenne.

LDAP-hakemiston rakenne on puumainen ja koostuu LDAP-hakemistomerkinnöistä, kuten kuvassa 17 esitetyssä esimerkissä nähdään. Ylimpänä on hakemistopuun juuri. Hakemistopuu on jäsennelty puusegmentteihin, jotka etenevät yksittäisiin lehtiin asti. Juuressa kuvatut DC (Domain Component) -komponentit ovat LDAP-palvelimen verkkotunnuksen osat. DC-komponentit luetaan oikealta vasemmalle. Esimerkiksi juuren DC-komponentista ”com” etsitään komponentti ”example”.

Varmenteella tunnistettavassa nimessä (DN, Distinguished Name) on vähintään yleinen nimi (CN, Common Name), mutta voi olla muitakin tietoja, kuten maakoodi tai organisaatio. Näiden tietojen perusteella syntyy hakemistoon puurakenne. Esimerkiksi kuvan 17 DigiCert juurivarmenteen (RootCA) tunnistettava nimi on:

```
/CN=DigiCert RootCA/OU=www.digicert.com/O=Digicert Inc/C=US
```

Jos pelkästään tätä juurivarmennetta haettaisiin lokaalista LDAP-arkistosta, löytyisi se LDAP-osoitteesta:

```
ldap://localhost:389/cn=DigiCert%20RootCA,dc=example,dc=com
```

Esimerkistä 7 nähdään, että tämän kaltainen haku palauttaisi kaikki tallennetut tiedot CN-komponentissa määritetystä juurivarmentajasta, kuten juurivarmenteen, hylkäyslistan ja tiedon tyypit.

```
$ curl ldap://localhost:389/cn=TestRootCA,dc=example,dc=com
DN: CN=TestRootCA,dc=example,dc=com
            objectClass: certificationAuthority-V2
            objectClass: top
            objectClass: applicationProcess
            objectClass: certificationAuthority

            cACertificate;binary:: MIIDVTCCAj2gAwIBAgIUdqBK7rwe6L...
            cn: TestRootCA

            certificateRevocationList;binary:: MIIBrDCBlQIBATANBg...
```

Esimerkki 7. LDAP-vastausviestin rakenne.

LDAP-vastausviesti toimitetaan BER (Basic Encoding Rules) -binaariformaatissa, mutta esimerkissä 7 curl-komentorivityökalu on muuntanut tämän valmiiksi luettavampaan muotoon. Koska kaikkia tietoja ei välttämättä tarvita, voidaan esimerkiksi tehdä haku esimerkin 7 hylkäyslistaan (certificateRevocationList):

```
ldap://localhost:389/cn=TestRootCA,dc=example,dc=com?certificateRevocationList;binary
```

Tämä osoite voitaisiin sitten toimittaa x509v3-varmenteiden CRL DP (Certificate Revocation Distribution point) -laajennuksessa.

Oletusarvoisesti LDAP-palvelimen ja asiakkaan välisiä viestejä ei salata, mikä tarkoittaa, että palvelimen ja asiakkaan verkkoliikennettä voidaan seurata. PKI:n hylkäyslistat ja varmenteet ovat julkista tietoa, joiden aitoudet voidaan varmentaa allekirjoituksilla. Tästä syystä ei liikenteen salaamiselle ole tyypillisesti PKI-

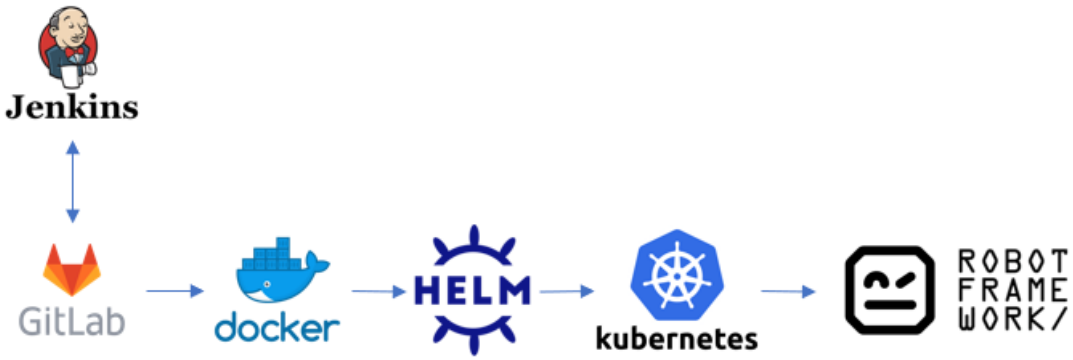
toteutuksissa tarvetta, mutta jos jostain syystä olisi, voidaan käyttää LDAP-protokollaa TLS-protokollan päällä. Tämä protokolla tunnetaan paremmin nimellä LDAPS (Secure LDAP).

Koska EJBCA ei tue julkaisijan määritysten asettamista sen komentotulkilla, jouduttiin nämä viemään suoraan tietokantaan XML-formaatissa. Käytännössä tämä tapahtui tekemällä tarvittavat julkaisijan määritykset graafisessa käyttöliittymässä ja kaappaamalla tietokannasta nämä tiedot. Tiedot pystyttiin sitten dumppaamaan MariaDB-tietokantaan tämän käynnistyessä käyttäen Kubernetes-asennusmääritystä.

Jotta CA-yksiköt pystyttiin julkaisemaan LDAP-hakemistoon, piti nämä yksiköt vielä määrittää käyttämään tehtyjä julkaisijan asetuksia. Tarvittavat komennot lisättiin tehtyyn Shell-komentosarjaan, joka suoritetaan EJBCA-kontin käynnistyessä.

# 5 CA-palvelimen liittäminen osaksi testiautomaatiota

Viimeisenä tarpeena insinöörityössä oli integroida toteutettu CA-palvelu nykyiseen sertifikaattihallintapalvelun CI/CD (Continious Integration/Continious Delivery) -järjestelmään. Ennen päivitetyn Docker-levykuvan julkaisemista tulisi tämän toiminta testata kohdeympäristössä kaikkien riippuvuuksien kanssa. CI/CD-järjestelmä voidaan toteuttaa usealla eri tavalla, mutta tässä tapauksessa on se toteutettu käyttäen Jenkins-automaatiopalvelinta, joka suorittaa E2E-testit Kubernetes-klusteriympäristössä käyttäen Robot Framework -testiautomaatiokehystä. Robot Framework on tieto- ja avainsanavetoinen testiautomaatiokehys, jonka kehitys on alkanut Pekka Laukkasen (nykyään Pekka Klärck) diplomityöstä (Laukkanen 2006).




Kuva 18. Yksinkertaistettu kohdeympäristön Docker-konttitestiautomaation kulku.

Kuvassa 18 kuvatussa Docker-konttien testiautomaation kulussa Jenkins-palvelin saa ilmoituksen Git-palvelimelta, josta hakee Docker-kontin asennusmääritykset ja suorittaa uuden levykuvan rakentamisen Docker-ohjelmistolla. Uusi levykuva asetetaan käytettäväksi Helm-asennusmäärityksissä ja asennetaan kaikkien muiden riippuvuuksien kanssa Kubernetes-klusteriin. Eli rakennettu CA-palvelu tulisi myös tässä asentaa muiden mukana. Helm-asennuksissa myös asennetaan kapseli, joka toimii loppukomponenttina sertifikaattihallintapalvelulle käyttäen Robot-kehystä.

CA-palvelu on jo tässä vaiheessa paketoitu valmiiksi ja toimitettu pakettivarastoon. Pakettivarasto on virtuaalinen tietoverkon kautta käytettävä hakemisto, minne ohjelmistopaketteja voidaan tallettaa jaettavaksi. Esimerkiksi Helm-paketoitu Kubernetes-asennusmääritys voidaan kätevästi ottaa käyttöön pakettivarastosta kahdella Helm-komennolla:

```
helm repo add varaston-nimi http://osoite-varastoon.com/varaston-nimi
helm install varaston-nimi/asennus-kaavion-nimi
```

Nämä komennot voitiin lisätä Jenkins-automaation suorittamaan Helm-kaavioasennusten Shell-komentosarjaan, jolloin CA-palvelu käynnistetään yhdessä kaikkien muiden riippuvuuksien kanssa.

Jotta CMP-operaatioiden Robot-testejä pystyttäisiin helposti suorittamaan useammalla eri CMP-palvelimena toimivalla CA-ohjelmistolla, oli tarve toteuttaa Robot-testien suoritukseen tapa, jolla CMP-palvelimen määritykset voitaisiin asettaa. Joustavaksi tavaksi osoittautui Robot-kehyksen variablefile-komentoriviargumentti, joka mahdollistaa CMP-konfiguraatioiden suorittamisen Python-moduulista. Määritetylle Python-tiedostolle on mahdollista antaa argumentteja toteuttamalla ”get_variables”-niminen funktio. Jos kyseinen funktio löytyy, Robot-kehys kutsuu sitä ja odottaa paluuarvoksi hajautustaulua.

```
def get_variables(ip_or_pod_prefix, server_port):
    server_ip = None
    if is_valid_ip(ip_or_pod_prefx):
          server_ip = ip_or_pod_prefix
    else:
          server_ip = get_pod_ip(ip_or_pod_prefix)

    return {
        'server_ip': server_ip,
        'server_port': server_port,
    }
```

Esimerkki 8. Yksinkertaistettu esimerkki joustavasta CMP-määritysten toteuttamisesta.

Esimerkissä 8 on kuvattu yksinkertaisesti Robot-kehyksen variablefile-komentoriviargumentin mahdollisuuksia. Tässä tapauksessa pystyttiin antamaan määritellylle Python-moduulille argumenttina joko palvelimen IP-osoite tai Kubernetes-kapselin (Pod) nimen etuliite, jolla kapselin IP-osoite klusterissa pystyttiin selvittämään. Koska kaikkia Python-moduuleja pystytään käyttämään, mahdollistaa tämä joustavan testimuuttujien määrittämisen eri ympäristöjä varten. Esimerkin 8 kaltainen tiedosto voitaisiin esimerkiksi nimetä asianmukaisesti ja määritellä robot-suorituskomennossa:

```
$ robot --variablefile cmp_settings.py:${IP_ADDR}:${PORT} tests/
```

Näin argumenttina annettujen muuttujien arvot ovat muuttujina kaikkien Robot-testien käytettävissä. Tämän kaltainen Robot-komentoriviargumentti lisättiin Robot-testien Shell-komentosarjaan, millä määriteltiin kaikki tarvittavat asetukset

CMP-protokollapyyntöihin, mikä mahdollistaa joustavan CMP-testien suorituksen useammalla eri CMP-palvelimella.

Koska CI-testiautomaation suoritukset hidastavat toteutuksien valmistumista, olisi näiden hyvä olla mahdollisimman nopeita. EJBCA-kapselin käyttöönotossa todettiin, että kapselin noin viiden minuutin käynnistymisaika ei kuulosta optimaaliselta sen käyttötarkoitukseen ja tutkittiin, pystytäänkö valmiiseen PrimeKeyn toimittamaan EJBCA Docker -levykuvaan tekemään parannuksia. Uusi levykuva voitaisiin myös itse rakentaa EJBCA-lähdekoodista optimaalisemmaksi testiautomaation käyttötarpeisiin, mutta siihen käytettävä aika ei todennäköisesti vastaa siitä saatavaa hyötyä.

Käynnistysnopeuden parantamiseksi tutkittiin CI-putkiston suoritusnopeuksia kapselin lokitiedostoista. Suorituksista valittiin satunnaisesti 15 asennusta, joista otettiin ylös EJBCA-kapseliin käynnistykseen kuluneet ajat. Taulukossa 1 esitetyistä havainnosta huomataan, että EJBCA-ohjelmiston alustukseen ja konfiguraatiokomentojen suorittamiseen menee suhteellisesti huomattavasti enemmän aikaa verrattuna sovelluspalvelimen käynnistykseen.

Taulukko 1. EJBCA-kapselin käynnistyksen keskimääräiset suoritusajat.

| **Operaatio** | **Keskimääräinen suoritusaika sekunnin tarkkuudella (s)** |
|---|---|
| MariaDB-kontin käynnistyksen odotus ja tietokannan alustus | 8 |
| Sovelluspalvelimen käynnistys | 81 |
| EJBCA-ohjelmiston alustukset, kuten saapuvan yhteyden TLS-asetukset, juurivarmenteiden luonti, CMP-konfiguraatioiden ja LDAP-konfiguraatioiden alustus. | 202 |

Koska CMP-protokolla toimii HTTP-protokollan päällä ja LDAP-protokolla suoraan TCP-protokollan päällä, ei TLS-yhteyttä tarvita testitapauksiin ja voitaisiin poistaa TLS-alustukset käynnistyksen nopeuttamiseksi. TLS-alustuksen ohittamiseksi päällekirjoitettiin sovelluskontissa sijaitseva Shell-komentosarja, mutta jätettiin mahdollisuus ottaa TLS-yhteyden käyttöön Helm-argumenttina annetulla ympäristömuuttujalla. Pois jättäminen nopeutti huomattavasti EJBCA-kontin suoritusvalmiutta, mutta suoritettavien konfiguraatioiden Shell-komentosarjassa on vielä parannettavaa.

EJBCA-komentotulkin komentojen suorittaminen vie aikaa, eikä kaikkien komentojen paluuta tarvitse odottaa ennen kuin edetään seuraavaan. Komentojen suoritusnopeutta pystyttiin parantamaan suorittamalla taustalla komennot, joiden suorituksesta muut eivät olleet riippuvaisia. Näitä olivat esimerkiksi CA-sertifikaattien ja hylkäyslistojen julkaisu LDAP-palvelimelle ja CMP-määritysten tuonti. Unixissa taustaprosessi voidaan suorittaa lisäämällä ampersandi (&)

Shell-komennon perään, jolloin prosessi haarautuu ja tehtäviä suoritetaan rinnakkaisessa prosessissa asynkronisesti. Näin saatiin EJBCA-alustuksien suorituksiin kuluva aika tiputettua keskimääräiseen 114 sekuntiin.

# 6 Yhteenveto

Insinöörityössä tutkittiin julkisen avaimen infrastruktuuria ja sen protokollia. Tutkimuksen pohjalta pyrittiin selvittämään, löytyisikö valmista avoimen lähdekoodin CA (Certificate Authority) -ohjelmistoa, joka tukisi sertifikaattihallintapalvelun tarpeita sen lopputesteissä ja mitä järjestelmän käyttöönottaminen virtuaalisessa klusteriympäristössä vaatii. Työssä rakennettiin Kubernetes-klusteriympäristöön EJBCA-ohjelmistosta CMP-palvelimena toimiva CA, joka olisi mahdollisimman helposti määriteltävä testitapauksia varten ja tutkittiin sen mahdollisuuksia ohjelmistotestauksen näkökulmasta.

PKI, eli julkisen avaimen infrastruktuuri, on digitaalisten varmenteiden hallintaan tarkoitettu rakenne, joka koostuu useista eri komponenteista ja käytännöistä. Näitä komponentteja ovat muun muassa CA (Certificate Authority), joka myöntää ja ylläpitää digitaalisia varmenteita luotetuille käyttäjille. Digitaalinen varmenne sisältää epäsymmetrisellä salausalgoritmilla luodun julkiseen käyttöön tarkoitetun salausavaimen, tietoja tämän epäsymmetrisen avainparin haltijasta sekä sertifiointiviranomaisen allekirjoituksen. Digitaalisia varmenteita käyttävät laitteet, ohjelmistot ja palvelut niiden luotettavuuden todistamiseksi**.**

Koska CMP-palvelimena toimiva CA:n suoritusympäristö tulisi olla Kubernetes-klusterissa, oli insinöörityössä tarpeen tutkia tämän toimintaa. Kubernetes on CNCF:n (Cloud Native Computing Foundation) ylläpitämä Docker-ohjelmistokonttien automatisoituun hallintaan tarkoitettu hallintaohjelmisto. Samalla selvitettiin Helm-pakettijärjestelmän toimintaa ja huomattiin, kuinka kätevää ja monipuolisia mahdollisuuksia Helm-pakettijärjestelmä tuo ohjelmistojen asennuksiin Kubernetes-klusteriympäristössä.

Insinöörityössä selvitettiin, mitä ohjelmistoja tarvitaan CA-palvelun rakentamiseen. Näistä päädyttiin testaaman EJBCA-ohjelmistoa. EJBCA on PKI-ohjelmisto, joka toteuttaa suurimman osan PKI:n komponenteista. Yksi EJBCA-asennus voi toimia samanaikaisesti sertifiointi- ja rekisteröintiviranomaisena, ja voi sisältää useita CA-instansseja. Nämä ominaisuudet mahdollistavat virtuaalisesti minkä tahansa PKI-rakenteen implementoinnin. EJBCA-ohjelmiston selvittämisessä huomattiin, ettei se toteuta LDAP-hakemistoprotokollaa, koska tyypillisesti nämä ovat erillisiä ohjelmistoja. Varmenteiden hylkäyslistojen tallentamista varten on LDAP-hakemisto välttämätön, sillä kohteena oleva sertifikaattihallintapalvelu tukee vain LDAP-protokollaa hylkäyslistojen lataamista varten. Tästä syystä etsittiin sopivaa toteutusta hakemistolle. LDAP-hakemistoksi valikoitui OpenDJ-ohjelmisto, joka on avoimen lähdekoodin LDAPv3-yhteensopiva hakemistopalvelu.

EJBCA-ohjelmiston konfiguroinnissa tehtiin profiileja digitaalisille varmenteille ja tutkittiin, miten EJBCA-ohjelmistossa voidaan tehdä luottamusketjuja luomalla useita sertifiointiviranomaisia. Konfiguroinnissa tutkittiin myös CMP (Certificate Management Protocol) -protokollaa, joka on digitaalisten varmenteiden hallintaan tarkoitettu viestintäprotokolla. Tutkimus painottui CMPv2-protokollaan, joka on 3GPP-standardissa määritelty rajoitetumpi profiili CMP-protokollasta. Työssä selvitettiin CMP-protokollaviestien rakennetta ja toimintaa sekä millaisia määrityksiä EJBCA-ohjelmistoon piti tehdä, jotta se toimisi sertifikaattihallintapalvelun kanssa. Koska EJBCA-ohjelmisto ei mahdollista useiden eri salausmenetelmien hyväksymistä samassa CMP-profiilissa, jouduttiin tekemään eri CMP-profiilit varmenteen rekisteröitymispyynnölle ja päivityspyynnöille.

Työssä tutkittiin LDAP-hallintaprotokollaa käyttävän julkaisijan toimintaa ja LDAP-hakemistorakennetta sekä määritettiin LDAP-palvelimen asetukset EJBCA-ohjelmistolle. LDAP-hakemistoprotokolla on Internet-hakemistojen lukemiseen ja muokkaamiseen. LDAP-hakemistorakenne on puumainen ja sen tarkoituksena on mahdollistaa tietojen etsiminen organisaatioista, henkilöistä ja muista resursseista, kuten tiedostoista ja laitteista. LDAP-hakemistoprotokolla

tarjoaa siis optimaalisen säilytystavan julkisille digitaalisten varmenteiden ja varmenteiden hylkäyslistoille.

Tehdyille EJBCA-ohjelmiston konfiguraatioille etsittiin ratkaisua, jolla ne olisi valmiiksi käytössä asennuksen jälkeen. Suurin osan konfiguraatioista oli mahdollista tallentaa konfiguraatiotiedostoihin, jotka pystyttiin asettamaan Kubernetes-kapselin asennusmäärityksissä. Kaikille tämä ei ollut mahdollista, vaan jouduttiin ottamaan tietokannasta konfiguraatioiden raakadata talteen ja lisäämään nämä suoraan tietokantaan ennen EJBCA-sovelluskontin käynnistämistä.

Toteutettu Helm-paketoitu Kubernetes-asennusmääritys julkaistiin yksityiseen pakettivarastoon, josta CA-palvelin on helposti käyttöönotettavissa sertifikaattihallintapalvelun CI (Continous Integration) -putkessa. Jotta CMP-määritykset voitiin helposti asettaa Robot-kehykselle, kehitettiin tapa, miten Robot-komentoriviltä pystyttiin CMP-määritykset asettamaan käyttäen Robot-kehyksen variablefile-komentoriviargumenttia.

Insinöörityön lopussa analysoitiin EJBCA-kapselin suorituskykyä ja tutkittiin, miten käynnistysnopeutta voitaisiin parantaa. Tähän ratkaisuna olisi toteuttaa tyhjästä oma EJBCA-levykuva sen lähdekoodista, mutta siihen käytettävä aika ja ylläpito ei todennäköisesti vastaa saavutettuja hyötyjä. Sen sijaan päällekirjoitettiin EJBCA-sovelluskontin alustuskomentoja ja poistettiin tarpeettomat alustukset testitapausten suorittamisessa. Lisäksi tutkittiin, miten voidaan hyödyntää Unixissa asynkronista suoritusta konfiguraatioiden alustamisessa.

Lopputuloksena saatiin versio EJBCA-kapselista, joka määrittää tarpeelliset konfiguraatiot CMP-protokollaan ja varmenteiden hylkäyslistojen testaamiseen heti käynnistyessä. Koska EJBCA-kapseli voidaan käynnistää helposti missä tahansa Kubernetes-jakelussa, esimerkiksi kehittäjän työkannettavalla, helpottaa se uusien testitapausten kehittämistä. Kehittäjä pystyy itse tekemään tarpeelliset konfiguroinnit EJBCA-ohjelmistoon testitapausta kehittäessä, eikä tuota lisätyötä palvelimen ylläpitäjälle, koska sellaista ei ole.

Työn tuloksia voidaan hyödyntää EJBCA-ohjelmiston käyttöönotossa Kuberenets-ympäristössä, varsinkin testiautomaation näkökulmasta. Työn digitaalisiin varmenteisiin ja niiden hallintaan keskittyvää teoriaosuutta on mahdollista hyödyntää sertifikaattihallintapalvelujen rakentamisessa.

# Lähteet

Adams C; Farrell S; Kuase T & Mononen T. 2005. Internet X.509 Public Key Infrastructure Certificate Management Protocol. Verkkoaineisto. Network Working Group. <https://www.ietf.org/rfc/rfc4210.txt>. Luettu 14.5.2021.

Chokani S; Ford W; Sabett R; Merrill C & Wu S. 2003. Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework. Verkkoaineisto. Network Working Group. <https://www.ietf.org/rfc/rfc3647.txt>. Luettu 14.5.2021.

Cooper D; Santesson S; Farrell S; Boyen S; Housley R & Polk W. 2008. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List Profile. Verkkoaineisto. Network Working Group. <https://www.ietf.org/rfc/rfc5280.txt>. Luettu 13.5.2021.

Deshpande, Raghavedra. 2019. YAML basics in Kubernetes. Verkkoaineisto. IBM. <https://developer.ibm.com/technologies/containers/tutorials/yaml-basics-and-usage-in-kubernetes/>. Luettu 13.5.2021.

Dierks T & Rescorla E. 2008. The Transport Layer Security Protocol Version 1.2. Verkkoaineisto. Network Working Group. <https://www.ietf.org/rfc/rfc5246.txt>. Luettu 13.5.2021.

DigiCert - Behind the Scenes of SSL Cryptography. Verkkoaineisto. DigiCert. <https://www.digicert.com/faq/ssl-cryptography.htm>. Luettu 2.5.2021.

Carter Eric. 2018. 2018 Docker usage report. Verkkoaineisto. Sysdig. <https://sysdig.com/blog/2018-docker-usage-report/>. Luettu 18.5.2021.

Google Cloud – Containers. Verkkoaineisto. Google. <https://cloud.google.com/containers>. Luettu 16.5.2021.

Laukkanen, Pekka. 2006. Data-Driven and Keyword-Driven Test Automation Frameworks. Espoo: Helsinki University of Technology.

Keyfactor – Certificate Chain of Trust. Verkkoaineisto. Keyfactor. <https://www.keyfactor.com/blog/certificate-chain-of-trust/>. Luettu 12.5.2012

Keyfactor - What is PKI. Verkkoaineisto. Keyfactor. <https://info.keyfactor.com/what-is-pki>. Luettu 12.5.2021

Kubernetes – What Is Kubernetes. Verkkoaineisto. Kubernetes. <https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/>. Luettu 16.5.2021.

Kubernetes – Components. Verkkoaineisto. Kubernetes. <https://kubernetes.io/docs/concepts/overview/components/>. Luettu 16.5.2021.

Kubernetes – Pods. Verkkoaineisto. Kubernetes. <https://kubernetes.io/docs/concepts/workloads/pods/>. Luettu 16.5.2021.

Oracle. Java™ Cryptography Architecture Standard Algorithm Name Documentation for Java Platform Standard Edition 7. Verkkoaineisto. < https://docs.oracle.com/javase/7/docs/technotes/guides/security/StandardNames.html#Mac>. Luettu 21.5.2021.

Ristić, Ivan. 2014. Bulletproof SSL and TLS. Lontoo: Feisty Duck Limited.

Scheiner, Bruce. 1996. Applied cryptography: protocols, algorithms, and source code in C, s.365-368. New York: Wailey.

Hickey, Shawn; Coulter, David; Jackobs, Mike; Farley, Patric & Satran, Michael. 2017. Intro to Certificates. Verkkoaineisto. Microsoft. <https://docs.microsoft.com/en-us/windows/uwp/security/certificates>. Luettu 13.5.2021.

Winnard, Keith; Von Dem Bussche, Martina; Choi Wai & Rossi David. 2016. Managing Digital Certificates across the Enterprise. IBM Redbooks.

Xenitellis, Symeon. 2000. The Open-Source PKI Book.

Zhou, Jianying; Kang, Meng-Chow; Bao, Feng & Pang, Hwee Hwa. 2005. Applied public key infrastructure: 4th International Workshop on Applied Public key Infrastructure. Washington: IOS Press.