Henri Tikkanen

# ÄLYKKÄÄN SOPIMUKSEN KEHITYS-PROSESSI HYPERLEDGER FABRIC -LOHKOKETJUYMPÄRISTÖÖN

Opinnäytetyö

Liiketalouden ammattikorkeakoulututkinto

Tietojenkäsittelyn koulutus

2022

Kaakkois-Suomen ammattikorkeakoulu

Kaakkois-Suomen ammattikorkeakoulu

| | |
|---|---|
| Tutkintonimike | Tradenomi (AMK) |
| Tekijä/Tekijät | Henri Tikkanen |
| Työn nimi | Älykkään sopimuksen kehitysprosessi Hyperledger Fabric -lohkoketjuympäristöön |
| Toimeksiantaja | Oy yritys Ab |
| Vuosi | 2022 |
| Sivut | 59 sivua |
| Työn ohjaaja(t) | Jari Kortelainen |

## TIIVISTELMÄ


Tämän opinnäytetyön tavoitteena oli tuottaa ensisijaisesti kokemusperäistä tietoa älykkään sopimuksen kehittämisestä *Hyperledger Fabric* -lohkoketjuympäristössä. Aihetta lähestyttiin ensin yleisen teorian kautta, josta siirryttiin tutkimaan älykkään sopimuksen kehitysprosessia kokonaisuudessaan, tarvittavia työvälineitä ja saatavilla olevaa lähdemateriaalia. Lopuksi toteutettiin älykäs sopimus hypoteettiseen finanssialan toimijaan liittyvän toimeksiannon perusteella ja implementointiin se lohkoketjuverkkoon.

Järjestelmään tutustumisen ja älykkään sopimuksen toteutuksen tukena käytettiin Hyperledger Fabricin tarjoamaa laajaa dokumentaatiota ja Fabric-testiverkkoa vapaasti ladattavine esimerkkeineen. Älykkään sopimuksen ohjelmointiin käytettiin *JavaScript*-kieltä. Fabric-testiverkon toimintaa kuvaavat kappaleet toteutettiin siten, että koko prosessi olisi toistettavissa esimerkkejä tarkasti seuraamalla. Varsinaisen älykkään sopimuksen kehitys toteutettiin puolestaan siten, että prosessi ja sen raportointi muodostivat sisällöllisen jatkumon aiemmille kappaleille.

Opinnäytetyössä havaittiin, että luotettavaa tietoa aiheesta oli kohtuullisen helposti saatavilla, mutta tieto oli kuitenkin verrattain kapea-alaista, painottuen Hyperledger-projektin julkaisemaan sisältöön. Myös suomenkielistä lähdemateriaalia löytyi suhteellisen niukasti. Merkittäviksi hidasteiksi Hyperledger Fabric -järjestelmän yleistymisen kannalta arveltiin mm. osaajapulaa ja järjestelmän profiloitumista vaativiin yritysmaailman toteutuksiin.

Lisäksi voitiin todeta, että onnistuttiin luomaan yksinkertainen konseptitodistus älykkään sopimuksen käyttämisestä määriteltyyn tarkoitukseen, joka toteutti sille asetetut vaatimukset tämän opinnäytetyön keskeisten tutkimustavoitteiden kontekstissa. Kun yhdisteltiin prosessin aikana kertynyttä teoriatietoa ja kokemuksia älykkään sopimuksen kehittämisestä Fabric-testiverkkoon, oli mahdollista tehdä tiettyjä johtopäätöksiä vastauksena tutkimusasetelmaan. Suhteessa muihin lohkoketjuteknologioihin, keskeisiksi Hyperledger Fabric -järjestelmän valintaa tukeviksi tekijöiksi nousivat yksityisyys ja yleinen tietoturvallisuus. Kun verrattiin perinteisiin järjestelmiin, todettiin lohkoketjuteknologiaan sisältyvän tiedon hajautuksen tarjoavan keskeisen turvallisuusedun kuvatun toimeksiannon tapauksessa, mutta toisaalta vaativan, että hajautus osataan käytännössä toteuttaa tarkoituksenmukaisella tavalla.



**Asiasanat:** älykäs sopimus, lohkoketjuteknologia, Hyperledger Fabric

South-Eastern Finland University of Applied Sciences

| | |
|---|---|
| Degree | Bachelor of Business Administration |
| Author (authors) | Henri Tikkanen |
| Thesis title | Development process of smart contracts to Hyperledger Fabric blockchain platform |
| Commissioned by | Oy Yritys Ab |
| Time | 2022 |
| Pages | 59 pages |
| Supervisor | Jari Kortelainen |

## ABSTRACT


The objective of this thesis was primarily to produce experiential information about the development process of smart contracts in the *Hyperledger Fabric* blockchain platform. The topic was first approached through general theory, followed by the development process of smart contracts studied as a whole, the tools required and the available source materials. Finally, a smart contract was implemented on the basis of an assignment related to a hypothetical financial operator and implemented in the blockchain network.

Extensive documentation provided by Hyperledger Fabric and the Fabric test network with freely downloadable examples were used to support the introduction of the system and the implementation of the smart contract. The *JavaScript* language was used to program the smart contract. The sections describing the operation of the Fabric test network were implemented in such a way that the whole process could be reproduced by closely following the examples. The process and reporting of developing the actual smart contract logically based on the previous paragraphs.

It was found in the thesis that reliable information on the topic was reasonably easily available, but the information was still relatively narrow as it was mainly published by the Hyperledger project itself. The source material in Finnish was also relatively scarce. As significant slowdowns in the spread of the Hyperledger Fabric system were considered to be e.g. a shortage of experts and focus on demanding enterprise-grade implementations.

In addition, it was found that a simple proof of concept for the use of smart contracts for a defined purpose was successfully created and it met the requirements set for it in the context of the main research objectives of this thesis. By combining theoretical knowledge and practical experience gained during the process, it was possible to draw conclusions in response to the research agenda. Compared to other blockchain platforms, privacy and general security appeared to be key factors supporting the choice of Hyperledger Fabric. Compared to traditional systems, the decentralization of data was found to provide a key security benefit in the case of the task described, but on the other hand it required that the decentralization be implemented in an appropriate manner.



**Keywords:** smart contract, blockchain technology, Hyperledger Fabric

**SISÄLLYS**

# 1 JOHDANTO

Tässä luvussa esitellään tausta ja henkilökohtainen motivaationi, jotka vaikuttivat tämän opinnäytetyön aihevalintaan. Lisäksi käydään läpi opinnäytetyön tutkimustavoitteet, tutkimuksen tarkempi rajaus sekä tutkimusongelma ja tutkimuskysymykset.

Aluksi kuitenkin muutama sana opinnäytetyön rakenteesta. Opinnäytetyön rakenne voidaan karkeasti jakaa kahteen laajempaan osioon, joista ensimmäinen käsittelee älykkäiden sopimuksien historiaa ja teoriaa empiirisen tutkimuksen tueksi. Tässä osiossa mm. luodaan katsaus lohkoketjuteknologiaan yleisesti sekä sen linkittymisestä älykkäisiin sopimuksiin ja tutustutaan *Hyperledger Fabricin* perusteisiin. Toisessa osiossa tutustutaan Hyperledger Fabric -ympäristöön käytännön esimerkkien kautta, jossa hyödynnetään Fabric-testiverkkoa ja Fabric-dokumentaation esimerkkejä. Lopuksi toteutetaan yksinkertainen älykkään sopimuksen malli, joka perustuu mahdolliseen tosielämän skenaarioon.

## 1.1 Tausta ja motivaatio

Olen seurannut läheltä lohkoketjuteknologian kehitystä noin 10 vuoden ajan, henkilökohtaisesta kiinnostuksesta aihetta kohtaan. Useimpien muiden aihetta pitkään seuranneiden tavoin tutustuin siihen kryptovaluuttojen kautta, joka oli lohkoketjuteknologian ensimmäisen käytännön sovellutus. Minua kiehtoi erityisesti lohkoketjuteknologian perusidea hajautetusta järjestelmästä, jossa luottamus perustuisi algoritmeille ja koneelliselle laskentateholle ihmisten sijaan (Nakamoto 2008). Kun tutustuin itse teknologiaan tarkemmin, aloin olemaan myös entistä vakuuttuneempi, että kyseessä todella on yksi merkittävimmistä teknisistä innovaatioista tietojenkäsittelyn lähihistoriassa. Viime vuosina olen yhä enemmän ollut kiinnostunut myös lohkoketjuteknologian muista sovellutuksista, kuten älykkäistä sopimuksista.

Keväällä 2021, avustaessani erään lohkoketjuteknologiaa käsittelevän opintojakson toteutuksessa Jyväskylän ammattikorkeakoulussa, tulin maininneeksi opintojakson vastuuopettaja *Tommi Hakalalle*, että olen harkinnut myös oman opinnäytetyöni tekemistä jostakin lohkoketjuteknologiaan liittyvästä aiheesta. Hän ehdotti minulle aiheeksi älykkäitä sopimuksia Hyperledger Fabric -

lohkoketjuverkossa, jonka parissa itse työskenteli. Itselläni ei ollut juuri minkäänlaista kosketuspintaa Hyperledger Fabric -järjestelmään ennen tätä. Minulle tutumpia aiheita lohkoketjuteknologian ja älykkäiden sopimusten saralla olisivat olleet mm. julkisen *Ethereum*-lohkoketjuverkon erilaiset sovellutukset, mukaan lukien älykkäät sopimukset (Ethereum 2022a). Koin kuitenkin, että näiden aiheiden objektiivinen tarkastelu olisi voinut olla minulle haasteellisempaa, johtuen läheisemmästä suhteestani niihin. Koin lisäksi, että Hyperledger Fabric ja ylipäätään yksityiset lohkoketjuverkot minulle ennestään vieraampana lohkoketjuteknologian osa-alueena, voisivat tarjota kiinnostavamman ja myös oman oppimiseni kannalta merkityksellisemmän tutkimusasetelman tälle opinnäytetyölle, joten päätin tarttua haasteeseen.

## 1.2 Tutkimustavoitteet, tutkimuksen rajaus ja tutkimusongelma

Tämän opinnäytetyön päätavoitteena on tutkia älykkäiden sopimuksien kehittämiseen käytettäviä työvälineitä ja työskentelymenetelmiä Hyperledger Fabric -ympäristössä sekä muodostaa konkreettinen malli kehitysympäristöstä. Prosessin lopputuloksena tulisi syntyä konsepti älykkään sopimuksen toteutuksesta sekä tämän prosessin havainnointi ja raportointi. Työn tavoitteena on myös saada yleistä tietoa älykkään sopimuksen kehitysprosessista sekä tutkia JavaScript-ohjelmointikielen käyttöä älykkään sopimuksen kehitysprosessissa Hyperledger Fabric -ympäristöön. Tavoitteena on lisäksi määritellä kehitysympäristön perusvaatimukset ja suositukset riittävällä tarkkuudella, jotta kuvattu prosessi olisi tämän opinnäytetyön lukijalle helposti toistettavissa.

Tässä opinnäytetyössä käytettävä keskeisin lähdemateriaali on Hyperledger Fabricin oma englanninkielinen dokumentaatio (kts. Hyperledger 2022b) sekä ladattavat Fabric-mallitiedostot (engl. *Fabric Samples*), joita hyödynnetään testiverkon pystyttämiseen sekä älykkään sopimuksen implementoimiseksi verkkoon (Hyperledger 2022d). Näin ollen tutkimuksesta saatavat tulokset rajautuvat myös voimakkaasti tämän käytettävissä olevan materiaalin perusteella. Muita lähteitä erityisesti aiheen teoreettiseen pohjustukseen ovat mm. tieteellisesti hyväksytyt julkaisut ja artikkelit sekä kehittäjien omat julkaisut.

Tutkimuksen ulkopuolelle on jätetty monia yleisesti lohkoketjuteknologiaan liittyviä keskeisiä konsepteja ja teknisiä yksityiskohtia, kuten erilaiset

konsensusmenetelmät tai tarkemmat kryptografiset menetelmät, joiden tarkkaa kuvaamista ei ole pidetty tutkimusaiheen ja valitun näkökulman kannalta tarkoituksenmukaisena. Tarkoitukseni tässä opinnäytetyössä on lähestyä älykkäitä sopimuksia mahdollisimman selkeästi itsenäisenä aihekokonaisuutenaan. Tämän vuoksi olen jättänyt pois myös kaikki tarpeettomat viittaukset mm. kryptovaluuttoihin ja muihin tämän opinnäytetyön varsinaisen aiheen näkökulmasta toissijaisempiin lohkoketjujen sovellutuksiin. Näitä aiheita ei kuitenkaan ole nykyisin luontevaa käsitellä täysin toisistaan irrallisina, sillä niillä on kiistatta joitain merkittäviä teknisiä ja historiallisia yhtymäkohtia, joita käsitellään tarkemmin luvussa 2.2. Älykkäiden sopimuksien juridisiin kysymyksiin ei myöskään oteta tässä työssä kantaa.

Edellä kuvattuja opinnäytetyön keskeisimpiä tavoitteita ja tutkimusongelmaa voidaan tarkentaa seuraavien tutkimuskysymysten avulla:

1. Millaista osaamista, pohjatietoa ja kehitystyökaluja vaaditaan käytännössä älykkäiden sopimuksien kehittämiseen Hyperledger Fabric -ympäristössä?
2. Kuinka hyvin tarvittavaa tietoa on saatavilla älykkään sopimuksen kehittämiseksi Hyperledger Fabric -ympäristöön?
3. Millaiseen tarkoitukseen olisi perusteltua käyttää Hyperledger Fabric -ympäristöä ja miten älykkäät sopimukset siinä poikkeavat älykkäistä sopimuksista muissa lohkoketjuympäristöissä?
4. Voisiko Hyperledger Fabric soveltua käytettäväksi toteutuksen yhteydessä kuvattuun tapaukseen ja millaisia etuja se voisi tarjota verrattuna esimerkiksi perinteisiin keskitettyihin järjestelmiin?

Varsinainen tutkimusongelma voitaisiin siis kiteyttää myös Hyperledger Fabric -järjestelmästä valmiiksi saatavilla olevan teoreettisen tiedon syventämiseksi omien kokeiluiden ja empiiristen havaintojen kautta. Hyperledger Fabricista ylläpidetään varsin kattavaa, säännöllisesti päivitettävää dokumentaatiota (kts. Hyperledger 2022b), mutta onko materiaali riittävä toimivan älykkään sopimuksen toteuttamiseksi verkkoon, ilman aiempaa mainittavaa kokemusta järjestelmästä? Entä löytyykö älykkään sopimuksen ja lohkoketjuteknologian käyttämiselle sellaisia perusteita, joiden pohjalta voisi esittää uskottavan argumentoinnin perinteisen keskitetyn tietorakenteen korvaamisesta näillä teknologioilla? Ja mikäli tällaisia perusteita löytyy, miksi tulisi valita juuri Hyperledger Fabricin tarjoama malli lohkoketjuteknologian toteutustavaksi?

# 2 ÄLYKKÄÄT SOPIMUKSET JA LOHKOKETJUTEKNOLOGIA

Tässä luvussa kartoitetaan älykkäiden sopimusten teoriaa yleisellä tasolla ja tehdään katsaus lohkoketjujen tietoturvaan sekä erityyppisiin lohkoketjuihin. Lisäksi tutustutaan Hyperledger Fabric -lohkoketjuympäristön perusarkkitehtuuriin, salaukseen ja älykkäisiin sopimuksiin Fabric-ympäristössä.

## 2.1 Älykkäiden sopimusten määritelmä ja historia

Älykkäät sopimukset (engl. *smart contracts)* ovat eri tieteenalojen rajoja ylittävä aihe, jota on tutkittu ja määritelty erityisesti oikeustieteiden ja tietojenkäsittelytieteiden toimesta. Aiheena älykkäät sopimukset on viime vuosien aikana noussut yhdeksi keskeisistä teemoista puhuttaessa lohkoketjuteknologiasta, mutta todellisuudessa älykkäät sopimukset eivät ole syntyneet lohkoketjuteknologian myötä (Lauslahti ym. 2016, 3). Älykkäiden sopimusten historian voidaan katsoa alkaneen jo 90-luvun alusta, kun kryptografiaan perehtynyt tietojenkäsittely- ja oikeustieteilijä Nick Szabo kuvaili älykkäiden sopimusten peruskonseptin julkaisemassaan artikkelissa *Smart Contracts*, osana digitaalisiin arvopapereihin, sähköiseen kaupankäyntiin ja sopimusoikeuden sähköistämiseen liittyviä julkaisujaan. (Kemmoe ym. 2020, 2.)

Laajemmin Szabo on kuitenkin esitellyt älykkäiden sopimusten ideaa artikkeleissaan *Smart Contracts: Building Blocks for Digital Markets* vuonna 1996 (Szabo 1996) ja *Formalizing and Securing Relationships on Public Networks* vuonna 1997 (Szabo 1997). Jälkimmäisessä julkaisussa hän myös esittelee klassisen ”myyntiautomaatti” -esimerkin, joka edelleen mainitaan yleisesti, kun pyritään tiivistämään älykkäiden sopimusten peruskonsepti helposti lähestyttävään muotoon. Siinä Szabo kuvailee perinteisen myyntiautomaatin toimintalogiikkaa ja rinnastaa sen älykkääseen sopimukseen; kuka tahansa, jolla on sopivia kolikoita, voi osallistua vaihtoon myyjän kanssa ja laitteen turvamekanismien ansiosta sen rikkomisesta aiheutuvan kustannuksen tulisi olla suurempi, kuin siitä koituvan hyödyn. Szabon mukaan ajatus älykkäistä sopimuksista menee kuitenkin paljon yksinkertaista ostotapahtumaa pidemmälle, sillä ne voisivat kytkeytyä kaikenlaiseen digitaalisesti hallinnoituun omaisuuteen lukemattomilla eri tavoilla. Szabo kuvailee, että älykkäät sopimukset viittaavat omaisuuteen yleensä dynaamisessa ja ennakoivasti pakotetussa muodossa, tarjoten perinteisiä sopimuksia selvästi vahvemman todentamisen ja

seurannan tason. Lauslahden ym. mukaan Szabon alkuperäinen määritelmä älykkäille sopimuksille oli ”koneellisesti luettava transaktioprotokolla, joka toteuttaa sopimuksessa ennalta määritellyt ehdot” (Lauslahti ym. 2016, 13). (Szabo 1997; Savelyev 2016, 7-16.)

Pohjimmiltaan älykkäät sopimukset voitaisiin edelleen kuvata Szabon alkuperäisen määritelmän mukaisesti, mutta niiden koko potentiaalin ymmärtämiseksi tämä määritelmä on melko riittämätön. Viimeistään lohkoketjuteknologian myötä määritelmään sisältyvänä oletuksena on myös voitu liittää älykkäiden sopimusten täysin autonominen toiminta, ilman tarvetta ns. luotetuille kolmansille osapuolille (engl. *Trusted Third Parties*), joita Szabo on itse kutsunut myös ”turva-aukoiksi” (Szabo 2001). Lauslahden ym. mukaan älykkäiden sopimusten määritelmää voitaisiin nykyisin tarkentaa siten, että ne ovat sopimusehtojen joukkoja, sisältäen myös ehtojen suorittamista rajoittavat transaktioprotokollat (Lauslahti ym. 2016, 13). Jatkossa tässä opinnäytetyössä myös viitataan ensisijaisesti tähän määritelmään älykkäitä sopimuksista puhuttaessa. On lisäksi syytä tarkentaa, että autonomisen toiminnan edellytyksenä älykkäät sopimukset kohdistuvat aina digitaalisten resurssien hallintaan tai fyysisten resurssien, joita voidaan suoraan ohjailla digitaalisesti.

## 2.2 Älykkäiden sopimusten kehitys lohkoketjuteknologian rinnalla

Szabon kehittäessä omaa konseptiaan älykkäistä sopimuksista 90-luvulla, ei kuitenkaan ollut olemassa vielä sellaista teknologista ympäristöä, joka olisi mahdollistanut älykkäiden sopimusten laajamittaisen hyödyntämisen (Lauslahti ym. 2016, 3). Tällainen ympäristö syntyi vasta yli 20 vuotta myöhemmin, kun vuonna 2009 Satoshi Nakamotona tunnettu pseudonyymi julkaisi oman ideansa virtuaalivaluutta *bitcoinista* ja sitä varten kehittämästään *Bitcoin*-protokollasta (Nakamoto 2008). Satoshi Nakamoton todellinen henkilöllisyys ei ole tiedossa, ja monien muiden ohella myös Nick Szaboa on useissa lähteissä spekuloitu Sakamotoksi, mutta hän on itse toistuvasti kieltänyt yhteyden (Frisby 2014).

Nakamoton esittelemä Bitcoin hyödynsi useita jo paljon aiemmin esiteltyjä ideoita ja niputti ne yhteen luodakseen uuden protokollan. Esimerkiksi *S. Haberin* ja *W. S. Stornettan* julkaisussaan *How to time-stamp a digital document*

esittelemää ratkaisua digitaalisten asiakirjojen aikaleimaukseen tiivistefunktioita hyödyntäen (Haber & Stornetta 1991), sekä *C. Dworkin ja M. Naorin* (Dwork & Naor 1993) alunperin roskapostien torjumiseen sähköpostiprotokollassa suunnattua kryptografista varmistusmenetelmää, voidaan pitää merkittävinä kiinnekohtina Nakamoton työlle (Rantala 2018). Suorana esikuvana Bitcoinin käyttämälle *Proof-of-Work*-konsensusmenetelmälle toimi puolestaan *Adam Backin* vuonna 1997 esittelemä *Hashcash*-algoritmi, jonka alkuperäinen tarkoitus oli niin ikään roskapostin sekä palvelunestohyökkäyksien torjuminen (Back 2002). Myös Szabon vuonna 1998 kehittämää *Bit gold* -protokollaa – jota ei koskaan julkaistu – voidaan pitää monella tavalla Bitcoinin varhaisena ”prototyyppinä” (Szabo 2008).

Nakamoton esittelemää protokollaa hajautetusta tietorakenteesta, jossa lohko on linkitetty toiseen lohkoon hajautusarvonsa (ts. *tiivistefunktion*) kautta, kutsutaan nykyään yleisemmällä tasolla *lohkoketjuteknologiaksi* (Nakamoto 2008). Voidaankin esittää, että Bitcoinissa mullistavinta ei niinkään ollut sen tekninen toteutus, kuin ajatus siitä, että luottamus järjestelmässä rakentuu täysin algoritmien varaan. Tietoturva-asiantuntija Mikko Hyppönen tiivistääkin lohkoketjuteknologian ”puhtaaksi matematiikaksi”, kuvaillessaan kryptovaluuttoja teoksessaan *Internet* (Hyppönen 2021). Samalla hän tulee kiteyttäneeksi myös sen, miksi lohkoketjuteknologia tarjosi täydellisen kasvualustan Szabon ideoimille älykkäille sopimuksille, ilman kolmansien osapuolien tai muiden inhimillisten tekijöiden väistämättä mukanaan tuomaa turvallisuusuhkaa.

Vaikka jo Bitcoin-protokolla mahdollistikin periaatteessa älykkäiden sopimuksien hyödyntämisen lohkoketjuverkossa, laajamittaisesti niitä ei koskaan ole käytetty Bitcoin-verkossa. Ainakin osittain tämä on johtunut siitä, että Bitcoinin käyttämä *Script*-ohjelmointikieli ei mahdollista loogisten toisto-operaatioiden hyödyntämistä (Kemmoe ym. 2020, 1). Tämä tietoinen valinta on tehty mm. palvelunestohyökkäyksien torjumiseksi Bitcoin-verkossa, mutta samalla se tekee myös älykkäiden sopimusten ohjelmoinnista ja käytöstä hankalaa (Lauslahti ym. 2016, 14), joskaan ei täysin mahdotonta (Wright 2018). Vasta vuonna 2013 Vitalik Buterin esittelemä ja vuonna 2015 julkaistu (Ethereum 2022c) Ethereum-protokolla mahdollisti älykkäiden sopimusten laajamittaisen käytön lohkoketjuverkossa ja teki niiden ohjelmoinnista verrattain helppoa (Ethereum 2022a). Ethereumissa käytetään erityisesti älykkäiden sopimusten

ohjelmointiin kehitettyä, oliopohjaista *Solidity*-ohjelmointikieltä, jonka syntaksi perustuu vahvasti JavaScriptiin (Ethereum 2022b).

## 2.3 Tietoturva lohkoketjuissa

Tässä luvussa lähestytään tietoturvaa lohkoketjuissa muutamien peruskäsitteiden kautta, pyrkien valaisemaan lohkoketjujen olennaisimpia eroavaisuuksia verrattuna perinteisempiin tietorakenteisiin ja niihin perustuviin järjestelmiin.

Yleensä keskeisinä perusedellytyksinä tiedon turvalliselle käsittelylle mainitaan kolme ominaisuutta, joiden mukaan on nimetty ns. *CIA-kolmio* (engl. *CIA-triad*); luottamuksellisuus (*confidentiality*), eheys (*integrity*) ja saatavuus (*availability*) (Oscarson 2003). Useimmat lohkoketjuteknologiaan perustuvat järjestelmät kykenevätkin toteuttamaan kolmion kahta viimeistä kirjainta vastaavat edellytykset perinteisiä järjestelmiä todennäköisemmin (ISACA 2017).

Tiedon *eheydellä* tarkoitetaan sitä, että tietoa ei ole muutettu luvatta tallennuksen, käsittelyn tai siirron aikana (NIST 2022c), eli se voitaisiin kiteyttää myös tiedon *muuttumattomuudeksi*. Tiedon eheys perustuu lohkoketjuissa tyypillisesti useille päällekkäisille kryptografisille menetelmille, jotka voivat vaihdella käytettävän lohkoketjuverkon mukaan. Kaikille niille yhteistä on joka tapauksessa tiedon eheyden varmentaminen menetelmällä, jota kutsutaan *konsensukseksi.* Konsensus voidaan toteuttaa käytännössä monilla eri tavoilla, joista edelleen yleisimpiä on Bitcoinin käyttämä alkuperäinen Proof-Of-Work-menetelmä (Aggarwal & Kumar 2021). Tärkeää ei kuitenkaan ole käytetty menetelmä, vaan se, että tiedon eheydestä voidaan saavuttaa jollakin hyväksi todetulla tavalla konsensus ja liittää uudet tiedot osaksi lohkoketjua. Manipuloidakseen mitä tahansa lohkoketjuun tallennettua tietoa, epärehellisten toimijoiden olisi kyettävä väärentämään koko lohkoketjun tiiviste sen alusta asti ja onnistuttava muodostamaan edelleen konsensus tiedon eheydestä (Kaasalainen 2018). Tämä voisi olla teoreettisesti mahdollista vain, mikäli epärehellisillä toimijoilla oli hallussaan enemmistö kaikista konsensuksen muodostamiseen käytettävistä resursseista, kuten laskentatehosta Proof-Of-Work-menetelmässä (Nakamoto 2008). Tätä pidetään useimpien lohkoketjujen kohdalla käytännössä lähes mahdottomana tai sen toteutus tulisi ainakin

saavutettavaan hyötyyn nähden hyökkääjälle liian kalliiksi. Taustalla on siis periaatteen tasolla sama ajatus, kuin luvussa 2.1 esitellyssä ”myyntiautomaatti” -esimerkissäkin.

Tiedon *saatavuudella* tarkoitetaan sitä, että tiedon luotettava saanti ja käyttö on jatkuvasti mahdollista, ilman viivytyksiä (NIST 2022a). Täydellistä saatavuutta on käytännössä mahdotonta koskaan saavuttaa keskitetyissä tietojärjestelmissä, sillä ilman tietoturvaloukkauksiakin järjestelmiä joudutaan säännöllisesti päivittämään ja huoltamaan. On myös selvää, että keskitetyt järjestelmät ovat alttiita lukuisille erilaisille uhkatekijöille, jotka voivat olla virtuaalisia tai fyysisiä ja tahallisesti aiheutettuja tai esimerkiksi tahattomia onnettomuuksia. Kun tieto lohkoketjussa hajautetaan lukuisille yksittäisille solmuille (engl. *nodes*), joista kaikki sisältävät täydellisen kopion koko lohkoketjusta, tieto on aina saatavissa, vaikka osa solmuista poistuisikin verkosta. Konkreettinen suoja saatavuuden häiriöitä vastaan saavutetaan kuitenkin vasta silloin, kun solmut ovat käytännössä riippumattomia toisistaan, eli niiden kaikkien toiminta ei ole sidottu esimerkiksi saman yrityksen sisäiseen verkkoon tai palveluntarjoajan runkoverkkoon. (Karaarslan & Konacakli 2020.)

Tiedon *luottamuksellisuus* on puolestaan lohkoketjuille hieman ongelmallisempi käsite, sillä tämän toteutumiselle keskeisenä edellytyksenä pidetään sitä, että vain valtuutetut tahot pääsevät tarkastelemaan ja käsittelemään luottamuksellista tietoa (NIST 2022b). Lisäksi henkilötietojen käsittelyä koskevien kansainvälisten asetusten mukaan henkilöillä itsellään tulisi aina olla mahdollisuus pyytää omien tietojensa täydellistä poistamista tietojärjestelmistä (Euroopan parlamentin ja neuvoston asetus (EU) 2016/679, 17. artikla). Tämä ei lohkoketjujen muuttumattoman luonteen vuoksi ole yleensä mahdollista. Näin ollen erityisesti julkisissa lohkoketjuissa luottamuksellisuus on toteutettava jollakin toisella tavalla, kuten tallentamalla varsinainen tieto lohkoketjun ulkopuolelle (*off-chain*), johon vain viitataan lohkoketjuun tallennetussa tiivisteessä (Hepp ym. 2018). Tämän opinnäytetyön tutkimuskohteena oleva Hyperledger Fabric pyrkii puolestaan ratkaisemaan luottamuksellisuuteen liittyvät kysymykset ensisijaisesti vahvalla käyttöoikeuksien kontrolloinnilla (Hyperledger 2020c).

## 2.4 Julkiset ja yksityiset lohkoketjut

Erityyppiset lohkoketjuteknologian sovellutukset voidaan karkeasti jaotella kahteen päätyyppiin; julkisiin (myös *käyttöoikeuskontroilloimaton*, engl. *permissionless*) ja yksityisiin (myös *käyttöoikeuskontrolloitu*, engl. *permissioned*) lohkoketjuihin. Julkisten ja yksityisten lohkoketjujen keskeisenä erona pidetään tyypillisesti sitä, kenelle sallitaan pääsy liittyä verkkoon, tallentaa tietoa lohkoketjuun ja tarkastella pääkirjan tapahtumia. Julkisiin lohkoketjuihin mielletään yleensä liittyvän myös yksityisiä lohkoketjuja vahvemman hajautuksen tason. Älykkäiden sopimusten näkökulmasta vahvan hajautuksen voidaan nähdä takaavan lisäksi sen, että sopimukset on mahdollista taltioida kaikkien sopimusosapuolten intressien kannalta neutraaliin ja luvussa 2.3 kuvatulla tavalla turvalliseen ympäristöön. Myös tiedon läpinäkyvyys mainitaan usein keskeisenä tekijänä julkisten lohkoketjujen puolesta. (Strehle 2020.)

Julkisten lohkoketjujen haasteina pidetään puolestaan rajoittamattoman hajautuksen myötä seuraavia skaalautuvuusongelmia sekä suuren läpinäkyvyyden kääntöpuolena heikkoa yksityisyyden tasoa, jonka vuoksi niihin ei käytännössä voida taltioida vahvasti luottamuksellista tietoa, kuten yrityssalaisuuksia tai henkilörekisteritietoja (Strehle 2020). Myös korkeat ja heikosti ennustettavat tapahtumakustannukset rajoittavat niiden käyttöä yritysmaailmassa (Jabbar ym. 2020). Näihin haasteisiin on mahdollista vastata yksityisillä lohkoketjuverkoilla – kuten kuten Hyperledger Fabric – joissa hajautusta voidaan rajoittaa ja osallistujien oikeuksia verkon hallintaan sekä sen sisältämään dataan voidaan tarkasti ja luotettavasti kontrolloida (Hyperledger 2020c).

Ainakin periaatteessa myös yksityinen lohkoketjuverkko voitaisiin toteuttaa menettämättä avoimen ympäristön olennaisia etuja, erityisesti läpinäkyvyyttä ja laajaa hajautuksen tasoa. Läpinäkyvyyden parantamiseksi ehdotettuja ratkaisuja olisivat esimerkiksi julkisen ja yksityisen lohkoketjun linkittäminen yhteen eräänlaisen sivuketjun (engl. *sidechain*) avulla, valikoitujen tietojen taltioiminen julkiseen lohkoketjuun ja *homomorfisen salauksen* sekä ns. n*ollatietotodistuksen* (engl. *zero-knowledge proof*) hyödyntäminen valikoitujen tietojen todentamisessa (Mitani & Otsuka 2020). Käytännössä homomorfisella salauksella tarkoitetaan kryptografista menetelmää, jossa salausta ei tarvitse välissä purkaa, ennen salatun tiedon käsittelyä (Fontaine 2007, 3).

Nollatietotodistuksella puolestaan viitataan menetelmään, jossa jokin alkuperäiseen tietoon liittyvä väite voidaan todistaa paljastamalla ainoastaan väitteen totuusarvo (Tani 2020). On myös syytä muistaa, että lohkoketjuverkon yksityisyys ei tarkoita automaattisesti avointa lohkoketjuverkkoa heikompaa hajautuksen tasoa, vaan ainoastaan hajautuksen rajoittamista. Käytännössä hajautuksen tason määrittävät mm. lohkoketjuverkon tekninen toteutustapa, käyttöympäristö- ja tarkoitus sekä toteutunut käyttöaste (Jaeseung ym. 2021).

## 2.5 Hyperledger Fabricin perusteet

Hyperledger on *Linux Foundationin* vuonna 2015 käynnistämä yhteistyöprojekti avoimeen koodiin perustuvien lohkoketjuympäristöjen sekä niitä tukevien kehitystyökalujen ja ohjelmistokirjastojen kehittämiseksi. Vuonna 2016 projektiin liittyivät myös *IBM*, *Intel* ja *SAP*. Myöhemmin mukaan on tullut myös useita muita merkittäviä teknologia, finanssi- ja ohjelmistoalan organisaatioita sekä akateemisia instituutioita. (Hyperledger 2020b.)

*Fabric* on Hyperledger-projektin alla kehitetty yksityinen lohkoketjuympäristö, jonka pääkehittäjänä on toiminut IBM. Se tarjoaa modulaariselle arkkitehtuurille perustuvan, käyttöoikeuskontrolloidun lohkoketjuympäristön erityisesti yrityskäyttöön, jossa vaaditaan avoimia lohkoketjuympäristöjä korkeampaa yksityisyyden tasoa, nopeutta ja skaalautuvuutta. (Hyperledger 2020b.)

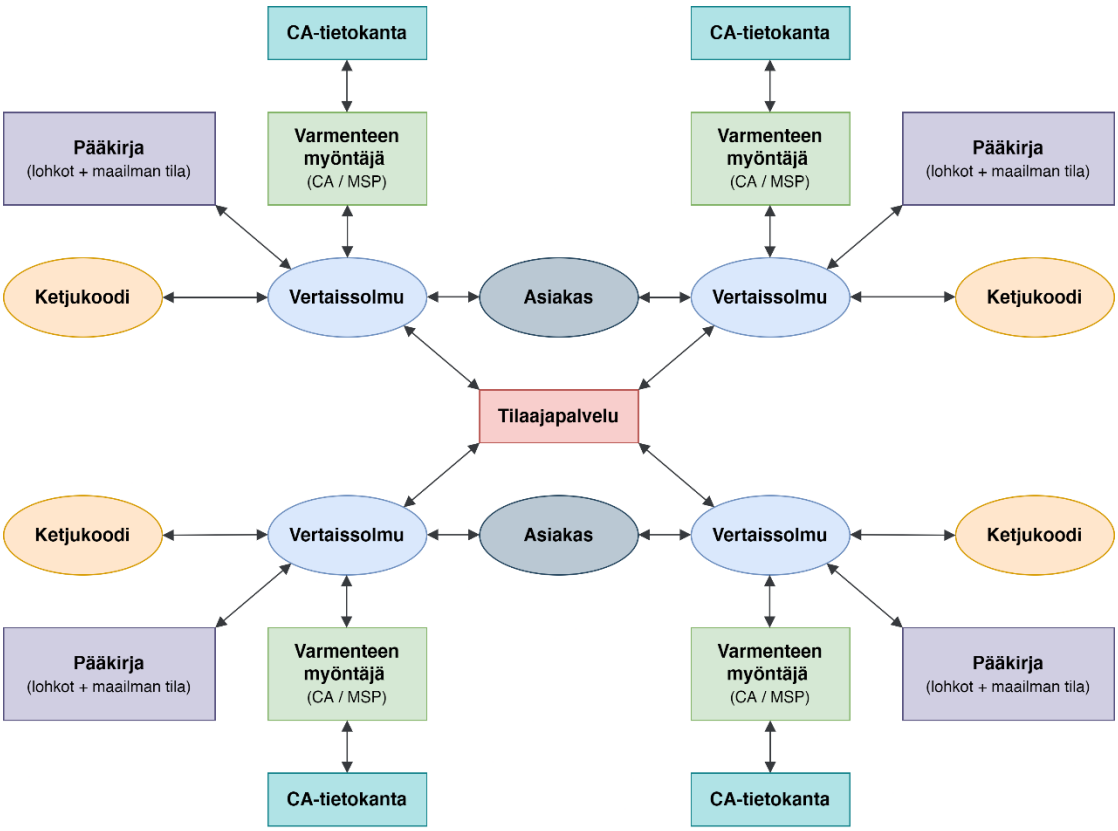
Hyperledger Fabric -verkko rakentuu pääosin samoista peruselementeistä, kuin useimmat muutkin lohkoketjuverkot, kuitenkin sisältäen myös joitain vain Fabricille ominaisia piirteitä ja käsitteitä. Seuraavat komponentit ovat keskeisiä Fabric-verkon kokonaisarkkitehtuurissa:

1. *pääkirja* (engl. *ledger*) sisältää kaksi toisiinsa linkittyvää erillistä osaa, jotka ovat *maailman tila* (engl. *World State)* ja lohkoketju. Maailman tilaa voi ajatella jatkuvassa muutoksessa olevana tietokantana, joka sisältää pääkirjan nykyiset arvot avain-arvo-pareina, jolloin nykyisten tietojen käsittely on suoraviivaista. Lohkoketjun voi puolestaan ajatella muuttumattomaksi tapahtumalokiksi kaikista nykyiseen maailman tilaan johtaneista tapahtumista, johon kaikki muutokset taltioidaan lohkoketjuteknologian perusperiaatteiden mukaisesti (Hyperledger 2022e).
2. *ketjukoodi* (engl. *chaincode*) on ohjelma, joka toteuttaa määrätyn käyttöliittymän ja liiketoimintalogiikan (engl. *business logic*), eli käytännössä sitä voidaan pitää älykkäänä sopimuksena (Salimitari ym. 2020).

3. *jäsenpalvelupalveluntarjoaja* (engl. *Membership Service Provider,* eli *MSP*) toimii eräänlaisena abstraktiona kaikista jäsenyystoiminnoista, mm. hallinnoiden varmenteita sekä toimien verkon jäsenten henkilöllisyyden ja roolien todentamiseksi. Yhtä lohkoketjuverkkoa voi hallinnoida yksi tai useampi jäsenpalvelupalveluntarjoaja modulaarisuuden sekä erilaisten jäsenyysstandardien yhteentoimivuuden varmistamiseksi (Salimitari ym. 2020).

Lisäksi verkon arkkitehtuuri voidaan jakaa seuraaviin keskeisiin toimijoihin:

1. *vertaissolmut* (engl. *peers* tai *peer nodes*), jotka mm. vastaavat pyyntöjen suorittamisesta ja vahvistamisesta, hallinnoivat pääkirjan tietoja ja älykkäitä sopimuksia, vuorovaikuttavat verkon sovellusten kanssa sekä validoivat uusia lohkoja (Salimitari ym. 2020)
2. *tilaajapalvelu* (engl. *orderer service* tai *orderer*) vastaanottaa kaikki pyynnöt (tapahtumat), luo niistä lohkon ja lähettää ne verkon vertaissolmuille vahvistettavaksi, sekä vastaanottaa hyväksytyt tapahtumat edelleen jatkokäsittelyä varten (Salimitari ym. 2020)
3. *varmenteen myöntäjät* (engl. *certificate authorities*) ovat verkon luottamia ”kolmansia osapuolia”, jotka myöntävät verkon käyttämät varmenteet jäsenpalveluntarjoajan määrittelemällä tavalla. Oletusarvoisesti ne ovat erillisiä palvelinohjelmistoja, joita kutsutaan nimellä *Fabric CA* (Hyperledger 2022d)
4. *asiakkaat* (engl. *clients*) ovat varsinaisen järjestelmän ulkopuolisia loppukäyttäjiä tai ohjelmia, jotka voivat kutsua älykkäitä sopimuksia tekemällä pyyntöjä verkkoon ohjelmointirajapinnan (*Fabric Gateway*) kautta (Hyperledger 2022c).




Kuva 1. Hyperledger Fabric lohkoketjuverkon perusarkkitehtuuri (mukaillen Baset ym. 2018)

Fabricin käyttämän modulaarisen arkkitehtuurin myötä tapahtumat erotetaan konsensuksesta ja ympäristöön voidaan liittää erilaisia lisäkomponentteja, kuten konsensus- ja jäsenpalveluita (Androulaki ym. 2018). Fabricissa konsensus muodostuu käytännössä monivaiheisen, tarpeiden mukaan räätälöitävissä olevan hyväksymispolitiikan kautta, jossa verkon hyväksytyt jäsenet muodostavat keskenään konsensuksen muutosten tilasta. Fabricissa konsensuksen muodostaminen ei näin ollen ole lukittuna mihinkään tiettyyn konsensusmenetelmään, mutta vakiintuneita menetelmiä ovat mm. *crash fault tolerant* (lyh. *CFT*) ja *byzantine fault tolerant* (lyh. *BFT*), joita voidaan hyödyntää valmiiden kirjastojen avulla (Barger ym. 2020; Hyperledger 2020d).

### 2.6 Salaus ja varmenteet Hyperledger Fabric -verkossa

Fabric käyttää tietojen salaukseen ns. *julkisen avaimen järjestelmää* (engl. *public key infrastructure*, lyh. *PKI*), joka on eräänlainen käytänteiden, prosessien ja tekniikoiden kokoelma, noudattaen kryptografian standardia *X.509*. PKI-järjestelmä kattaa myös Fabricin käyttämän *TLS*-salausprotokollan (engl. *Transport Layer Security*) sekä tiedon aitouden varmentamisen perustuen ns. luotettujen kolmansien osapuolien (engl. *Certificate Authority*, lyh. *CA*) varmentamille sertifikaateille. (Hyperledger 2019c; Hyperledger 2022h)

TLS on vanhempien *SSL-protokollien* seuraaja ja sitä käytetään nykyisin laajasti, mm. tavallisessa verkkoliikenteessä *HTTPS-protokollan* yhteydessä sekä sähköpostiliikenteen salaamiseen. TLS-protokollan on tarkoitus tarjota kokonaisvaltainen varmennus keskenään kommunikoivien kahden tai useamman sovelluksen välille, kattaen tiedon luottamuksellisuuden, eheyden sekä aitouden varmentamisen. Varsinainen protokolla koostuu kahdesta erillisestä tasosta; tiedon kulkua päätepisteiden välillä ohjailevasta *tallenneprotokollasta* ja tiedon todentamiseen sekä salaisen avaimen hallintaan keskittyvästä *kättelyprotokollasta*. Fabricissa vertaissolmu voi toimia TLS-asiakkaana muodostaessaan yhteyden toiseen vertaissolmuun tai tilaajaan sekä TLS-palvelimena, kun jokin toinen vertaissolmu, sovellus tai komentoliittymä muodostaa yhteyden siihen. (IBM 2021; Hyperledger 2019c)

Fabric CA on myös keskeinen ja itsessäänkin erittäin laaja aihekokonaisuus Fabricin kokonaisarkkitehtuurissa. Yleensä Hyperledger Fabricissa ns.

luotettuna kolmantena osapuolena on *CA-palvelin*, joka toimii jäsenpalvelun-tarjoajan (*MSP*) alaisuudessa. Tämä palvelin sisältää tietokannan tai *LDAP-protokollaan* perustuvan hakemistopalvelun, jossa varmenteita ylläpidetään. Lisäksi palvelimelle on asennettu Fabric SDK-ohjelmistopaketteja, jotka mahdollistavat vertaissolmujen kommunikoinnin CA-palvelimen kanssa, REST-tyyppisen ohjelmointirajapinnan välityksellä. CA-asiakkaat (engl. *CA-client*) ovat puolestaan tarkoitettu CA-palvelimien hallinnointiin. (Hyperledger 2022d.)

## 2.7 Älykkäät sopimukset Hyperledger Fabric -verkossa

Fabric-verkossa älykkäitä sopimuksia kutsutaan usein yleisnimellä ketjukoodi, jonka vuoksi tätä nimitystä tullaan käyttämään jatkossa myös tässä opinnäytetyössä laajasti. Tarkalleen ottaen nämä kaksi voidaan kuitenkin myös eriyttää toisistaan siten, että varsinainen älykäs sopimus määrittelee tapahtumien logiikan ja ohjailee objektien elinkaarta tietorakenteen sisällä, kun puolestaan ketjukoodi on älykkään sopimuksen sisältämä paketti, joka implementoidaan verkkoon. Ketjukoodi on siis mahdollista käsittää *luokkana*, joka voi sisältää älykkäiden sopimuksien *joukkoja*. (Hyperledger 2020j).

Ketjukoodissa käytännössä määritellään, miten verkon loppukäyttäjät voivat vuorovaikuttaa lohkoketjun pääkirjan kanssa, sekä miten tietoa käsitellään. Fabricissa ketjukoodin ohjelmointiin voidaan käyttää yleisiä ohjelmointikieliä, kuten *Go*, *Java* sekä JavaScript (*Node.js*). Fabricin luottamusmalli perustuu useille allekirjoituksille (engl. *signatures*), joten yksittäinen verkon jäsen ei voi muokata tai käyttää ketjukoodia ilman muiden hyväksyntää. Oletuksena suurimman osan jäsenistä tulee hyväksyä muutokset, ennen kuin ne voidaan ottaa käyttöön, mutta ketjukoodin hyväksymiskäytäntöjä voidaan myös muokata. (Hyperledger 2020e).

Organisaatioiden on asennettava ketjukoodi samaan kanavaan liitetyille vertaisilleen voidakseen vahvistaa tapahtumia tai tehdä kyselyitä pääkirjasta. Tämän jälkeen ketjukoodi voidaan ottaa käyttöön kanavalle ja käyttää ketjukoodin sisältämiä älykkäitä sopimuksia resurssien luomiseen tai päivittämiseen kanavan pääkirjassa. Ketjukoodin käyttöönotossa noudatetaan prosesseja, joista käytetään yleisnimitystä *elinkaaritoiminnot* (engl. *lifecycle methods*). Elinkaaren hyväksymiskäytännöt on eriytetty itse ketjukoodin

hyväksymiskäytännöistä ja riippumatta ketjukoodille määritetyistä vaatimuksista, enemmistön kanavan jäsenistä on joka tapauksessa hyväksyttävä ketjukoodin määritelmä elinkaarikäytäntöjen mukaisesti. (Hyperledger 2020e).

## 2.8 Älykkäiden sopimusten käyttökohteet ja lohkoketjuteknologian yleistyminen

Älykkäät sopimukset voivat olla lohkoketjuun tallennettuja ohjelmia, joita käytetään jonkin käyttöliittymän välityksellä tai ne voivat olla täysin autonomisesti toimivia sovelluksia, jotka suorittavat niille asetetut tehtävät tiettyjen ehtojen toteutuessa, jolloin käyttöliittymän sijaan tehtävät laukaistaan automatisoitujen laukaisinten (engl. *triggers)* avulla. Usein älykkäät sopimukset ovat yhteydessä esimerkiksi julkiseen verkkoon rajapintojen kautta, jolloin niille voidaan välittää erilaisia tietoja tai pyyntöjä tehtävien käynnistämiseksi. Suosituimpien julkisten lohkoketjuverkkojen kohdalla kehitys onkin viime vuosina kohdistunut erityisesti älykkäiden sopimusten hyödyntämiseen erilaisten hajautettujen sovellusten taustalla, jotka pyrkivät automatisoimaan ja varmentamaan esimerkiksi digitaalisiin omistuksiin liittyviä prosesseja. (Kemmoe ym. 2020).

Tällaisista voidaan mainita hyvänä esimerkkinä ns. *DeFi* (eli *Decentralized Finance*) -palvelut, jotka hyödyntävät älykkäitä sopimuksia hajautettujen finanssipalveluiden taustalla (Zetzsche ym. 2020) sekä ns. *NFT* (eli *Non-fungible token*) -palvelut, jotka hyödyntävät älykkäitä sopimuksia digitaalisten omistussuhteiden hallintaan esimerkiksi digitaalisen taiteen markkinoilla (Wang ym. 2021). Nämä palvelut perustuvat yleensä Ethereumille tai muille julkisille ns. toisen tai kolmannen sukupolven lohkoketjuverkoille, jotka on suunniteltu hyödyntämään tehokkaasti älykkäitä sopimuksia erityyppisten digitaalisten sitoumusten hallintaan (Xu ym. 2019).

Kuten jo edellä on todettu, Hyperledger Fabric -lohkoketjuverkon kehityksen painopiste on ollut alusta alkaen julkisia lohkoketjuverkkoja korkeampi turvallisuuden ja yksityisyyden taso, kuitenkaan menettämättä teknologian keskeisiä etuja verrattuna perinteiseen keskitettyyn palvelinarkkitehtuuriin. Tämän myötä myös ensisijaiset käyttökohteet painottuvat niihin yritysmaailman toteutuksiin, joissa näiden tekijöiden merkitys korostuu (Hyperledger 2020b).

Lohkoketjuteknologiaan yleisellä tasolla liittyvässä julkisessa keskustelussa toistuu edelleen usein argumentti, jonka mukaan se tarjoaa paljon lupauksia, mutta kuitenkin todistettavasti toteutuneet, konkreettiset hyödyt ovat olleet verrattain vähäisiä, eivätkä ne useinkaan ratkaise mitään merkittäviä ja arkipäiväisiä ongelmia (Honkanen 2017, 45). Suomessa tätä käsitystä on lisäksi tukenut mm. Valtioneuvoston julkaisema raportti lohkoketjuteknologian hyödyntämiseen julkisen vallan näkökulmasta (Valtiovarainministeriö 2019). Hyperledger-projektin omilla kotisivuilla esitellään kuitenkin lukuisia yksityiskohtaisia kuvauksia Fabriciin ja muihin projektin alla kehitettyihin järjestelmiin perustuvista, toteutuneista ratkaisuista, jotka pyrkivät ratkaisemaan hyvin konkreettisia tosielämän ongelmia. Fabricin osalta tällaisia ovat mm. seuraavat:

1. globaali metalli- ja kaivosteollisuuden toimitusketjujen hallintajärjestelmä (Hyperledger 2021a)
2. maarekisterin hallintajärjestelmä Abu Dhabin kaupungille (Hyperledger 2021b)
3. kaupankäyntialusta pk-yritysten, tuottajien ja rahoituslaitosten välille (Hyperledger 2021c)
4. ratkaisu reseptilääkkeiden jakeluketjun turvallisuuden kehittämiseksi Yhdysvaltain elintarvike- ja lääkevirastolle (Hyperledger 2020a)
5. Amazon-tyyppinen käytettyjen lentokoneen osien kauppapaikka Honeywell Aerospacelle (Hyperledger 2019a)
6. ruuan jakeluketjun hallintajärjestelmä Wallmartille (Hyperledger 2019b).

Suomalaisessa yhteiskunnassa lohkoketjuteknologiaan suhtaudutaan yleisesti melko varauksellisesti edelleen, vaikka sen mahdollisuudet kuitenkin tunnistetaan jo useilla toimialoilla. Esimerkiksi logistiikka- ja finanssialoilla monilla suomalaisilla yrityksillä on kehitteillä olevia hankkeita lohkoketjuteknologiaan liittyen, vaikkakin toteutuneet ratkaisut ovat toistaiseksi olleet vähäisiä (Nyyssölä & Paczkowski 2020). Yhtenä esimerkkinä lupaavasta käyttökohteesta lohkoketjuteknologialle Suomessakin on mainittu jo viime vuosikymmenen puolella lääkkeiden jakeluketju (Jäntti 2018), mutta konkreettisia avauksia tälläkään saralla ei vielä ole nähty. Myös Hyperledger Fabriciin pohjautuvia ratkaisuja kuitenkin kehitetään aktiivisesti Suomessa suurien toimijoiden, kuten Telia Cygaten (Telia Cygate 2022) sekä CGI:n toimesta (CGI 2020).

Yhdeksi lohkoketjuteknologian yleistymistä jarruttavaksi tekijäksi *Petri Honkanen* nosti jo vuonna 2017 kansainvälisen osaajapulan tutkimuksessaan *Lohkoketjuteknologian lupaus* (Honkanen 2017, 42–43). Viime vuosina suomalaisessa mediakeskustelussa on ollut runsaasti esillä myös yleinen osaajapula

IT-alalla ja tilanteen ennustetaan työnantajien kannalta edelleen jatkuvasti vaikeutuvan (Yle 2017). Työmarkkinoille suuntautunut verkkoyhteisöpalvelu LinkedIn puolestaan raportoi lohkoketjuteknologiaan ja kryptovaluuttoihin liittyvien työnhakuilmoitusten määrän kasvaneen pelkästään Yhdysvalloissa 395 % vuodesta 2020 vuoteen 2021 (LinkedIn 2022). Kun lisäksi otetaan huomioon lohkoketjuteknologiaan erikoistuneen koulutuksen puute kansainvälisesti (Themistocleous ym. 2020), voidaan todennäköisin perustein olettaa osaajapulan jarruttavan lohkoketjuteknologian yleistymistä myös lähitulevaisuudessa.

# 3 TESTIVERKKO JA KEHITYSYMPÄRISTÖ

Tässä luvussa tutustutaan ensin testiverkon pystytykseen melko tarkasti Fabricin dokumentaatiosta (kts. Hyperledger 2022b) löytyviä esimerkkejä seuraten, mutta kuitenkin omia havaintojani tästä prosessista painottaen. Tarkoituksena myös olisi, että seuraamalla tarkasti tässä kuvattuja esimerkkejä, prosessin toistaminen onnistuisi ilman laajoja ennakkotietoja aiheesta.

Testiverkon pystytyksen jälkeen perehdytään varsinaisen kehitysympäristön määrittelyyn, jolle onnistunut testiympäristön pystytys toimii luontevana lähtökohtana. Niin itse Fabricin lähdekoodia kuin myös dokumentoinnin versiota päivitetään tiiviisti, joten on syytä huomioida, että tässä esitetyt esimerkit ja huomiot dokumentaatiosta koskevat lähtökohtaisesti vain nykyistä versiota (tätä kirjoittaessa 2.4.3). Järjestelmän perustoimintoihin tulee kuitenkin harvoin olennaisia muutoksia ja mahdolliset muutosten aiheuttamat ongelmat on yleensä mahdollista paikallistaa ja ratkaista tutustumalla dokumentoinnin uusimpaan versioon. Virallista dokumentaatiota päivitetään yleensä ainakin alaversiopäivitysten (engl. *minor*) yhteydessä ja kuka tahansa voi ehdottaa siihen muutoksia projektin GitHub -tilin kautta. (Hyperledger 2021d).

## 3.1 Hyperledger Fabric -testiverkko

Hyperledger Fabric -testiverkolla ja varsinaisella kehitysympäristöllä on monelta osin samanlaiset perusvaatimukset. Ennen varsinaisen kehitysympäristön määrittelyä on kuitenkin erittäin suositeltavaa tutustua valmiiksi luotuihin Fabric-malleihin testiverkossa. Testiverkossa voidaan myös varmistaa, että

verkon toiminnan perusvaatimukset täyttyvät järjestelmän ja tarvittavien kehitystyökalujen osalta. (Hyperledger 2021d).

Hyperledger Fabric testiverkko voidaan pystyttää *Windows*, *Mac* tai *Linux* -ympäristöön ja vaadittavat määrittelyt sekä esivalmistelut poikkeavat joiltakin osin riippuen isäntäjärjestelmästä, käytettävien perustyökalujen ollessa kuitenkin samoja. Testiverkko sisältää Fabric-malleja (engl. *Fabric samples*), joiden tarkoitus on ennen kaikkea auttaa ymmärtämään verkon perustoimintoja ja rakennetta. Testiverkkoon luodaan mukana tulevien valmiiden komentosarjojen (engl. *script*) avulla toiminnan havainnollistamiseen tarvittavat kaksi vertaisorganisaatiota ja tilaajaorganisaatio. Testiverkko on tarkoitettu käytettäväksi vain suljetussa ympäristössä testaustarkoituksessa, joten osapuolien todennukseen ei ole tarkoituksenmukaista soveltaa jäsenpalveluntarjoajan myöntämiä TLS-sertifikaatteja. Tarpeettoman kompleksisuuden välttämiseksi todennus suoritetaan näin ollen juuritasolla. Salaus ja varmenteet Fabric -ympäristössä on kuvattu tarkemmin luvussa 2.6. Testiverkon mukana toimitetaan myös yksinkertainen ketjukoodin malli, jonka avulla voidaan havainnollistaa resurssien (engl. *assets*) siirtämisen peruskonseptia verkossa. (Hyperledger 2022f).

### 3.2 Suositeltavat kehitystyökalut testiverkkoon

Tässä luvussa käydään läpi tarvittavat työkalut ja niiden vaatimukset, perusmäärittelyt sekä käyttötarkoitukset Linux-ympäristössä (*Ubuntu* 20.04). Suositeltavat työkalut testiverkon pystytykseen ovat seuraavat:

1. *Docker* (Linuxille myös erikseen *Composer*)
2. *Git*
3. *cURL* (tulee myös Git:n, Dockerin ja uusimpien Windows-versioiden mukana, mutta Fabricin kehittäjät suosittelevat kuitenkin lataamaan uusimman version suoraan cURL-projektin sivustolta)
4. Komentokehoitetyökalu (esim. *Linux Bash*). (Hyperledger 2022f.)

Docker on järjestelmätason virtualisoinnille perustuva, Linux-pohjainen kokonaisuus, jolla voidaan hallita konteiksi (engl. *containers*) kutsuttuja ohjelmistopaketteja. Kokonaisuuteen kuuluu itse Docker-sovelluksen lisäksi myös mm. *Composer*-niminen työkalu, jota käytetään useiden pakettien hallinnoimiseksi ja ajamiseksi Dockerissa samanaikaisesti. Tarvittaessa ladataan ja asennetaan Dockerin sekä Composerin viimeisimmät versiot kehittäjän ohjeiden mukaan (Docker 2022a).

On myös syytä huomioida, että erityisesti käytettäessä *Docker Desktop* -sovellusta Windows-ympäristössä, vaaditaan melko paljon laitteistoresursseja. Sovelluksen käyttöön suositellaan mm. RAM-muistia vähintään 4GB. Myös virtualisointitukea vaaditaan laite- sekä ohjelmistotasolla (Docker 2022b). On myös melko yleistä, että ennen Dockerin käyttöä tuki virtualisoinnille pitää erikseen kytkeä päälle käyttöjärjestelmän asetuksista tai laitteiston BIOS-asetuksista (Docker 2022c).

Git on versiohallinnan perustyökalu, jota käytetään jossakin muodossaan lähes kaikessa ohjelmistokehityksessä. Jatkokehitystä ajatellen tässä vaiheessa olisi myös suositeltavaa luoda esim. *GitHub*-tili ja linkittää se *Git*:iin, mikäli tiliä ei vielä ole olemassa. Tarvittaessa ladataan ja asennetaan *Git*:n viimeisin versiot kehittäjän ohjeiden mukaan (Git 2022).

cURL on komentokehoitteella toimiva työkalu datan siirtämiseen URL-syntaksin avulla erilaisia verkkoprotokollia käyttäen. Tarvittaessa ladataan ja asennetaan *cURL*:n viimeisin versiot kehittäjän ohjeiden mukaan (cURL 2022).

### 3.3 Testiverkon vaatimukset ja esivalmistelu

Tässä esimerkissä käytetään Ubuntu-pohjaista Linux-ympäristöä, mutta myös eri Windows-versiot sekä Mac-ympäristöt ovat mahdollisia. On kuitenkin syytä huomioida, että Hyperledger Fabric ja monet käytettävät kehitystyökalut ovat aluksi kehitetty Linux-ympäristöön ja muissa ympäristöissä saattaa ilmetä erilaisia yhteensopivuusongelmia, joita todennäköisesti Linux-ympäristössä ei havaita. Verkossa saatavilla oleva yhteisön tuki on myös yleensä kattavampaa ja luotettavampaa Linux-ympäristöä käytettäessä. Tämä koskee erityisesti itse Fabric-ympäristöä sekä Dockeria. Myös kaikki Fabricin dokumentaatiossa esitetyt komennot ovat *Unix*-komentoja. Linuxia voidaan ajaa myös Windows-järjestelmän sisällä käyttäen erillistä virtualisointiohjelmaa.

Testiverkossa käytetään Fabric-esimerkkipakettia, joka sisältää tarvittavan ohjelmakoodin, komennot ja esimerkkimääritykset toimivan testiverkon pystyttämiseksi. Tässä yhteydessä käytämme JavaScript-kielistä versiota ketjukoodista, joka sisältyy oletuksena olevan Go-kielisen version tavoin pakettiin. Go-

kieltä käytetään useissa verkon sisäisissä komponenteissa ja mallisovelluksissa, mutta Go-kielen asentaminen ei kuitenkaan ole tarpeen, mikäli sitä ei ole tarkoitus käyttää ketjukoodin kirjoittamiseksi. (Hyperledger 2022d.)

Aivan aluksi on syytä varmistaa, että Docker on käynnissä ja oikea Linux-käyttäjä on lisätty Docker-ryhmään:

```
$ sudo systemctl start docker
$ sudo usermod -a -G docker <käyttäjänimi>
```

Luodaan sitten mikä tahansa sopiva hakemistorakenne, jonne Fabric-mallit voidaan ladata. Fabric-dokumentaatio suosittelee käyttämään oletushakemistoa *HOME/go/src/github.com/<github_käyttäjätunnus>* Go -kielisille projekteille, mutta tämän noudattaminen ei ole tarpeen käytettäessä ainoastaan JavaScript-kielisiä ketjukoodeja. Kannattaa myös huomioida, että nykyiselle Linux-käyttäjälle on annettava täydet oikeudet luotavaan juurihakemistoon (tarvittaessa hakemiston omistajuuden voi vaihtaa nykyiselle käyttäjälle *chown* -komennolla). Luodaan siis kotihakemistoon uusi hakemisto *fabric* ja siirrytään hakemistoon:

```
$ mkdir -p $HOME/fabric
$ cd $HOME/fabric
```

Tämän jälkeen kloonataan viimeisin *hyperledger/fabric-samples*-pakettivarasto (engl. *repository*) GitHubista ja ladataan kyseiseen kansioon uusimmat versiot Docker-kuvatiedostoista sekä binääri- ja määrittelytiedostoista seuraavalla *curl*-komennolla (toimiva osoite tätä kirjoitteessa, joka saattaa muuttua):

```
$ curl -sSL https://bit.ly/2ysbOFE | bash -s
```

Paketti sisältää useita erilaisia työkaluja, joita tarvitaan Fabric-testiverkon käyttöön, kuten *orderer*, *peer*, *fabric-ca-client* ja *fabric-ca-server*. Kaikki tiedostot asennetaan automaattisesti *fabric-samples*-alakansioon. (Hyperledger 2022d.)

### 3.4 Testiverkon pystytys

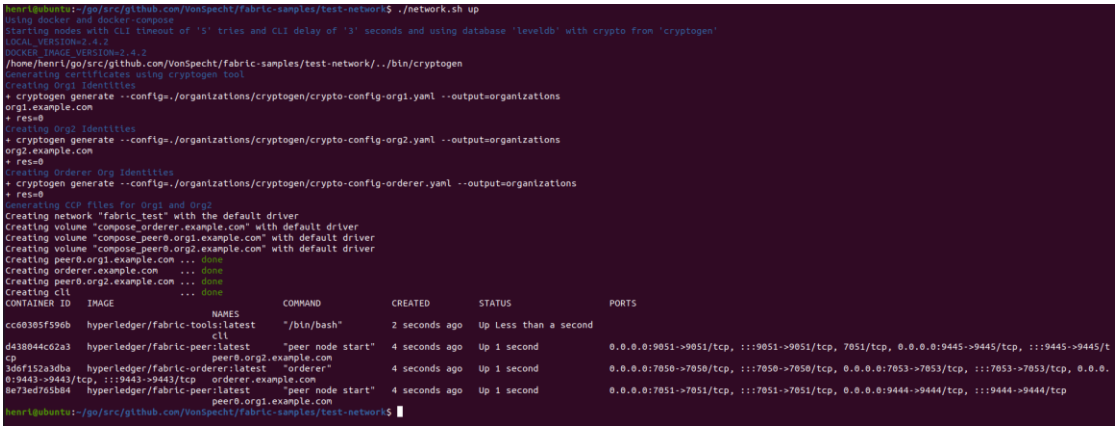
Verkon käynnistykseen käytetään *fabric-samples/test-network*-kansioon ladattua *network.sh*-komentosarjaa.

Aluksi on suositeltavaa aina varmistua, että mikään aiempi verkkoon liittyvä prosessi ei ole käynnissä ja varaa Docker-kontteja. Verkon alasajo tapahtuu seuraavasti*:*

*$ ./network.sh down*

Tämän jälkeen verkko voidaan käynnistää komennolla:

*$ ./network.sh up*




Kuva 2. Testiverkon käynnistys suoritettu onnistuneesti

Verkon käynnistys kestää hetken, kun komentosarja lataa Docker-kontit. Mikäli verkon käynnistys on sujunut ongelmitta, lopuksi tulisi näkyä neljä eri aktiivista Docker-konttia, joiden niminä (*names*-sarake) *cli*, *orderer.example.com*, *peer0.org2.example.com* ja *peer0.org1.example.com.* Aktiiviset kontit voidaan myös tarkistaa seuraavalla komennolla:

*$ docker ps -a*

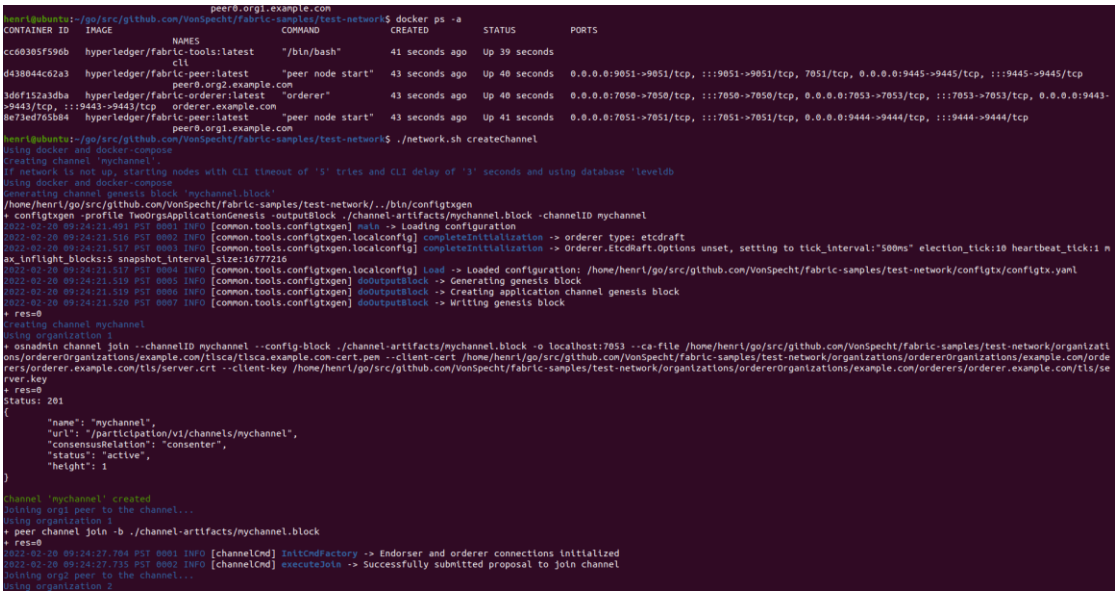
Seuraavaksi tarvitaan kanava (engl. *channel*), johon juuri luodut verkon jäsenet voivat liittyä. Käyttäen edelleen *network.sh* -komentosarjaa luodaan kanava oletusasetuksilla seuraavasti:

*$ ./network.sh createChannel*

Tämä käyttää kansion nimenä oletusta *mychannel* ja mikäli kanavan luominen on sujunut ongelmitta, nähdään lopuksi ilmoitus: *Channel 'mychannel' joined.*

Vaihtoehtoisesti voitaisiin myös suorittaa verkon käynnistys ja kanavan luonti yhdellä komennolla seuraavasti:

*$ ./network.sh up createChannel*




Kuva 3. Kanavan luonti suoritettu onnistuneesti

Mikäli kaikki on sujunut ongelmitta, tässä vaiheessa pitäisi olla Hyperledger Fabric -testiverkko toiminnassa, jossa on kaksi vertaisorganisaatiota (*Org1* ja *Org2*), tilaaja (*orderer*) sekä kanava verkon jäsenten väliseen vuorovaikutukseen. (Hyperledger 2022k.)

## 3.5 Älykkäät sopimukset testiverkossa

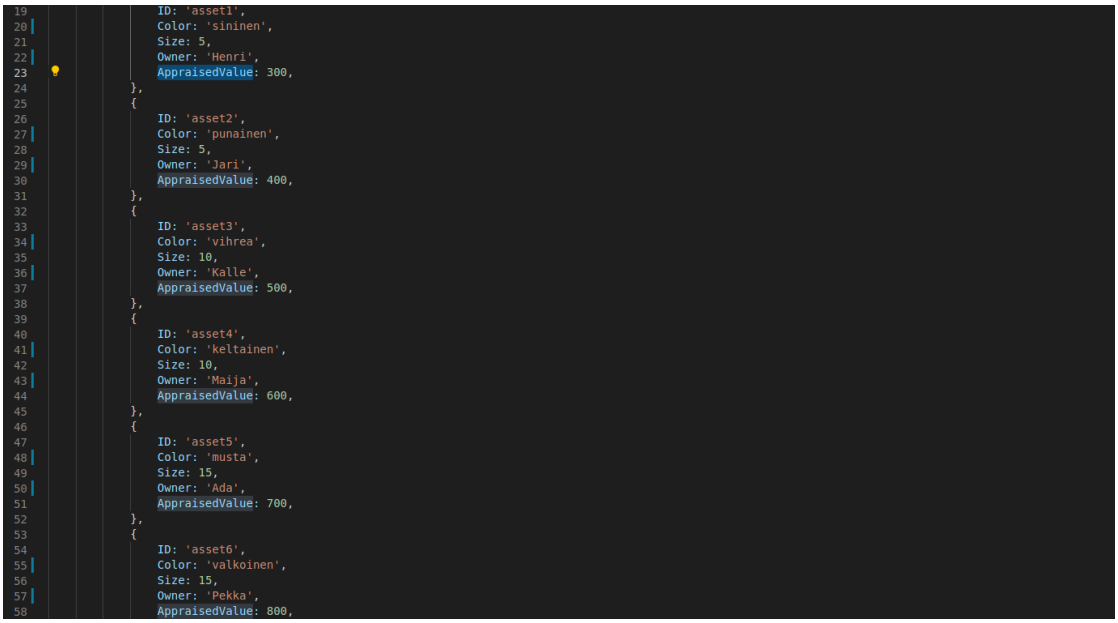
Kun testiverkon toiminnan perusedellytykset on luotu, voidaan alkaa luomaan sille merkitystä, eli siirtyä ketjukoodin käyttöönottoon verkossa. Poiketen Fabricin dokumentaatiosta, jossa käytetään oletusarvoisesti Go -kielistä versiota ketjukoodista, käytetään tässä siis JavaScript-versiota. Ketjukoodi sisältää etukäteen parametrisoidut muuttujien arvot, kuten käsiteltävien resurssien ominaisuudet sekä omistajat. Tässä yhteydessä muutetaan kuitenkin

muutamia valmiiksi parametrisoituja arvoja, jotta ketjukoodin perustoimintaa voitaisiin vielä paremmin havainnollistaa. (Hyperledger 2022k.)

Avataan seuraava tiedosto muokattavaksi teksti- tai IDE-editorissa:

*../fabric-samples/asset-tranfer-basic/chaincode-javascript/lib/assetTransfer.js*

Heti tiedoston alussa nähdään *InitialLedger*-funktio, jolle on määritelty muuttujiksi ominaisuudet *ID*, *Color*, *Size*, *Owner* ja *AppraisedValue*. Muutamme nyt ketjukoodin arvoja *Color* sekä *Owner* kaikille kuudelle resurssille (engl. *assets*). Uudet arvot voisivat periaatteessa olla mitä tahansa merkkijonoja, mutta käytetään tässä havainnollisuuden vuoksi ominaisuuden tyypille soveltuvia arvoja.

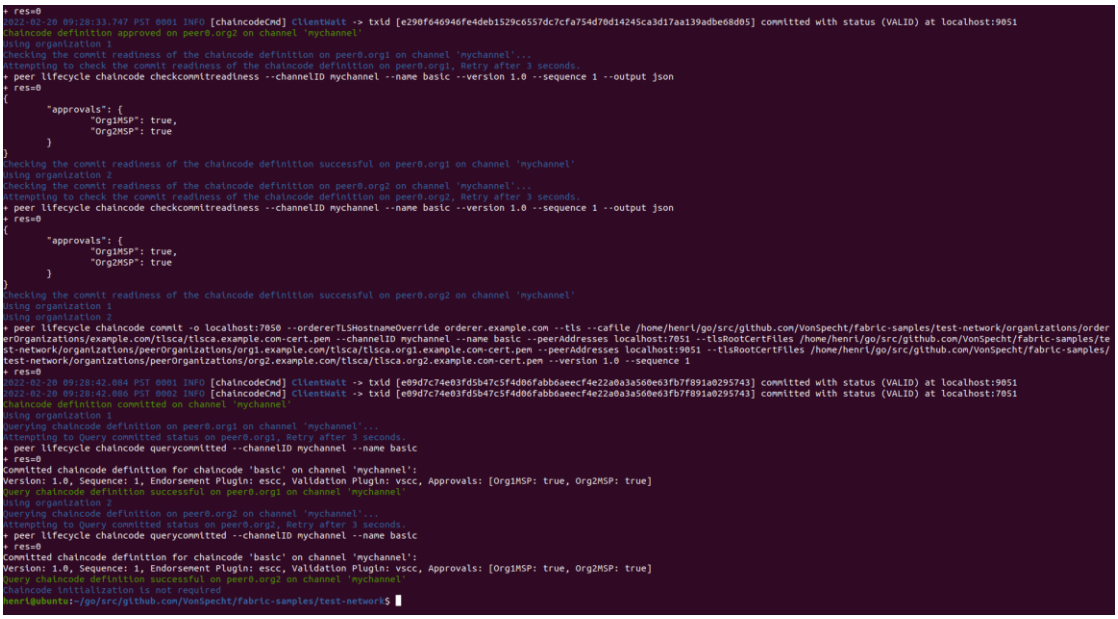



Kuva 4. Muutetaan resurssien alkuarvoja ketjukoodilta

Ketjukoodi on ensin paketoitava, jotta se voidaan asentaa ja ottaa käyttöön verkossa. Tämän jälkeen se tulee vielä hyväksyä verkon jäsenten toimesta hyväksymyskäytäntöjen mukaisesti. Käyttämämme *deployCC*-alakomento asentaa ketjukoodin määritetystä sijainnista molemmille vertaisorganisaatioille ja huolehtii sen käyttöönotosta kanavalle. Lisäksi se asentaa myös tarvittavat riippuvuudet määritellyn kielen mukaan. Seuraavalla komennolla suoritetaan siis käyttöönotto testipakettiin sisältyvän *basic*-nimisen ketjukoodin JavaScript-versiolle:

*$ ./network.sh deployCC -ccn basic -ccp ../asset-transfer-basic/chaincode-javascript -ccl javascript*

Mikäli ketjukoodin käynnistys sujui ongelmitta, voidaan ryhtyä valmistelemaan ensimmäistä siirtoa verkossa.




Kuva 5. Ketjukoodin käyttöönotto testiverkossa suoritettu onnistuneesti

Ensin määritellään kuitenkin seuraavat asetukset, jotta saadaan osoitettua testiverkon binääritiedostot oikeaan komentopolkuun sekä *FABRIC_CFG_PATH*-muuttujaan *core.yaml*-määrittelytiedosto:

*$ export PATH=${PWD}/../bin:$PATH*
*$ export FABRIC_CFG_PATH=$PWD/../config/*

Lisäksi määritellään jäsenpalveluntarjoajan (MSP) ympäristömuuttujat vertaisorganisaatiolle *Org1*, sisältäen *TLS-käytössä*-asetuksen, organisaation MSP-tunnisteen sekä sertifikaatin ja asetustiedon hakemistopolut:

*$ export CORE_PEER_TLS_ENABLED=true*
*$ export CORE_PEER_LOCALMSPID="Org1MSP"*
*$ export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt*
*$ export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp*
*$ export CORE_PEER_ADDRESS=localhost:7051*

Nyt voidaan kutsua ketjukoodin *InitLedger*-funktiota pääkirjan alustamiseksi ketjukoodille parametrisoiduilla arvoilla:

```
$ peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride
orderer.example.com --tls --cafile
"${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.c
om/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n basic --
peerAddresses localhost:7051 --tlsRootCertFiles
"${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.exampl
e.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRootCertFiles
"${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.exampl
e.com/tls/ca.crt" -c '{"function":"InitLedger","Args":[]}'
```

Mikäli toiminto suoritetaan onnistuneesti, pitäisi näkyä ilmoitus *Chaincode invoke successful*, tilakoodilla 200.

Seuraavaksi voidaan tehdä kysely vertaisorganisaation *Org1* puolesta, jolla saadaan näkyviin kaikki verkossa olevat resurssit, sisältäen niiden yksilölliset ominaisuudet, omistajat ja arvot.

```
$ peer chaincode query -C mychannel -n basic -c '{"Args":["GetAllAssets"]}'
```

Näkyviin pitäisi tulla *JSON*-muotoinen vastaus, jossa yksittäinen resurssi näyttää esimerkiksi tältä:

```
{"ID": "asset1", "color": "sininen", "size": 5, "owner": "Henri", "appraisedValue": 300}
```

Nyt voidaan kokeilla esimerkiksi omistajan vaihdosta kutsumalla ketjukoodia *invoke*-alakomennolla. Komento sisältää viittaukset molempien vertaisorganisaatioiden sertifikaattitiedostoihin, TLS-yhteyden varmentamiseksi tilaajaorganisaation toimesta:

```
$ peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orde-
rer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizati-
ons/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-
cert.pem" -C mychannel -n basic --peerAddresses localhost:7051 --tlsRootCertFiles
```

```
"${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt" -c '{"function":"TransferAsset","Args":["asset6","Rosvo"]}'
```

Onnistuneen siirron pitäisi jälleen palauttaa tilakoodi 200 ja resurssin *asset6* tulisi olla päivittynyt siten, että sen omistajaksi on merkitty nyt ”Rosvo”. Seuraavaksi voidaan kokeilla verkkoa myös vertaisorganisaation *Org2* näkökulmasta ja asettaa sille vastaavasti ympäristömuuttujat, kuten tehtiin aiemmin *Org1:n* kohdalla:

```
$ export CORE_PEER_TLS_ENABLED=true
$ export CORE_PEER_LOCALMSPID="Org2MSP"
$ export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt
$ export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp
$ export CORE_PEER_ADDRESS=localhost:9051
```

Nyt voidaan suorittaa *Org2*:n näkökulmasta uusi kysely *query*-alakomennolla kuten edellä, jolla vahvistetaan tapahtunut omistajan vaihdos. Mikäli kysely suoritettaisiin esimerkiksi resurssille *asset6*, tulisi muuttunut omistaja näkyä vastauksessa seuraavasti:

```
{"ID":"asset6","color":"white","size":15,"owner":"Rosvo","appraisedValue":800}
```

Vastaavasti voitaisiin muuttaa mitä tahansa muitakin resurssien ominaisuuksia. Tarkastelemalla älykkään sopimuksen lähdekoodia nähdään, että esimerkiksi funktiolla *UpdateAsset* voitaisiin päivittää haluttuja arvoja yksittäiseltä resurssilta muuttamalla aiemmin esiteltyä siirtopyyntöä esim. seuraavasti:

```
'{”function”:”UpdateAsset”,”Args”:[”asset1”,”punainen”,”99”,”Henri”,”99999”]}'
```

Tällöin resurssin *asset1* väriksi muutetaan *punainen*, kooksi *99*, omistajaksi *Henri* ja arvoksi *99999*. On syytä huomata, että annetulla komennolla kaikki arvot ilmoitetaan merkkijonoina sulkujen sisällä, vaikka alkuperäisellä

ketjukoodilla arvot on ilmoitettu lukuarvoina (*integer*). (Hyperledger 2020, Starting a chaincode on the channel.)

```
100     // UpdateAsset updates an existing asset in the world state with provided parameters.
101     async UpdateAsset(ctx, id, color, size, owner, appraisedValue) {
102         const exists = await this.AssetExists(ctx, id);
103         if (!exists) {
104             throw new Error(`The asset ${id} does not exist`);
105         }
106
107         // overwriting original asset with new asset
108         const updatedAsset = {
109             ID: id,
110             Color: color,
111             Size: size,
112             Owner: owner,
113             AppraisedValue: appraisedValue,
114         };
115         // we insert data in alphabetic order using 'json-stringify-deterministic' and 'sort-keys-recursive'
116         return ctx.stub.putState(id, Buffer.from(stringify(sortKeysRecursive(updatedAsset))));
117     }
```

Kuva 6. Fabric-testipaketista löytyvän malliohjelman *Assets Transfer Basic* funktio *UpdateAssets* (JavaScript-versio)

## 3.6 Kehitysympäristön valmistelu

Tässä kartoitetaan kevyesti, millaisia kehitystyökaluja vaaditaan tai on hyödyllistä käyttää ketjukoodin kehittämiseen Hyperledger Fabric -verkossa, käyttäen JavaScript-kieltä. Varsinaisen kehitysympäristön vaatimukset eivät olennaisesti poikkea luvussa 3.1 käsitellyistä testiverkon vaatimuksista ja testiverkon sujuva toiminta indikoi useimmissa tapauksissa myös ongelmatonta perustaa kehitysympäristölle. On syytä myös huomioida, että ”varsinaisella kehitysympäristöllä” ei tarkoiteta tuotantoympäristöä, vaan tässä toimitaan edelleen paikallisessa ympäristössä ja hyödynnetään *Fabric-samples*-paketin sisältöä. (Hyperledger 2021e.)

Aiemmin mainittujen työkalujen lisäksi varsinaiseen kehitystyöhön suositellaan ainakin seuraavia:

1. Jokin sopiva IDE-tekstieditori, kuten *Microsoft VS Code*
2. *Npm*, pakettienhallintatyökalu
3. *Logspout*, logitustyökalu.

IDE (*Integrated Development Environment*), eli ohjelmointiympäristö, sisältää sovelluskehitykseen optimoidun tekstieditorin lisäksi vähintään ohjelmointikielen kääntäjän, mutta yleensä lukuisia muitakin hyödyllisiä tai sujuvan kehitystyön kannalta välttämättömiäkin ominaisuuksia. Tällaisia voivat olla esimerkiksi ns. *debuggeri*, eli työkalu virheiden etsimiseen, sekä versionhallinta- ja

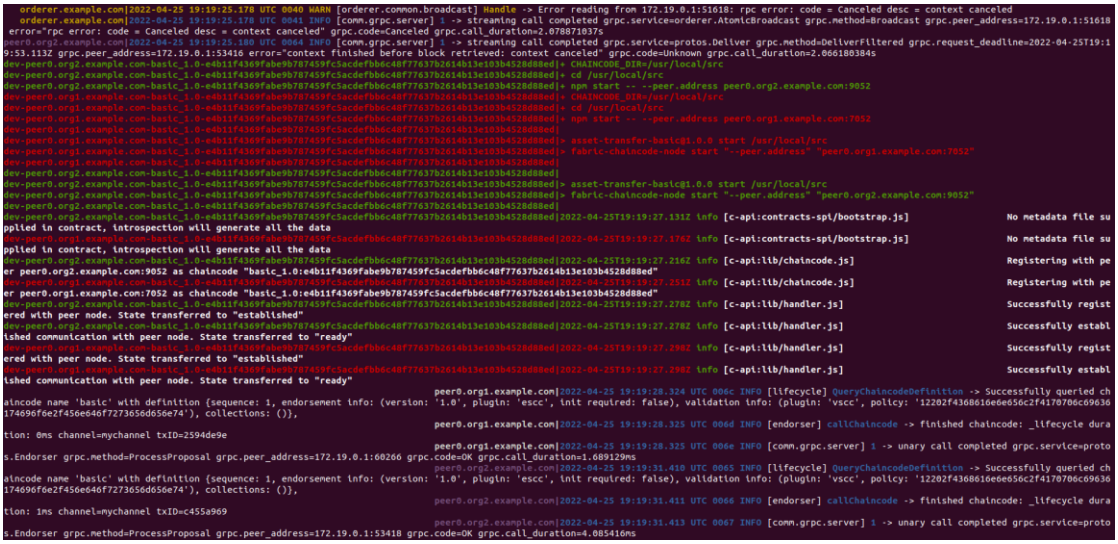
profilointityökalut. Myös eri ohjelmointikielille ja järjestelmille on kehitetty erikoistuneita IDE-editoreita, kuten Hyperledger Fabricille kehitetty *Chaincoder*, jonka beta-versio on tätä kirjoittaessa saatavissa Windows ja Mac-ympäristöille (Chaincoder 2022). Tässä yhteydessä käytetään kuitenkin *VS Coden* Linux-versiota, joka on helposti muokattava ja monipuolinen, laajasti kaikenlaisessa sovelluskehityksessä käytetty ohjelmointiympäristö (Visualstudio Code 2022).

Npm on JavaScript-pohjaisessa ohjelmistokehityksessä yleisesti käytetty pakettienhallintatyökalu ja ohjelmistorekisteri *Node.js*-pohjaisille projekteille (npmjs.com). Tässä yhteydessä Npm:ää käytetään ketjukoodin tarvitsemien riippuvuuksien asentamiseen ja hallinnointiin. Tarvittaessa ladataan ja asennetaan Npm:n viimeisin versio kehittäjän ohjeiden mukaan (Npm 2022).

Logspout on Docker-kontteihin liittyvien lokitietojen keräämiseen tarkoitettu työkalu, joka kerää tapahtumavirtaa useista erillisistä Docker-konteista yhteen paikkaan (Hyperledger 2022i). Ohjelma löytyy valmiiksi Fabric-mallien *test-network*-kansiosta ja sen voi käynnistää erilliseen komentokehoiteikkunaan seuraavalla komennolla, kun verkko on ensi käynnistetty:

```
$ ./monitordocker.sh fabric_test
```

Ohjelma aloittaa Docker-konttien seuraamisen välittömästi ja tulostaa ruudulle kaiken, mitä verkossa tapahtuu. Logspout ei ole välttämätön työkalu, mutta saattaa olla erittäin hyödyllinen ongelmien paikallistamiseen ketjukoodissa sekä suureksi avuksi haluttaessa oppia tuntemaan verkon toimintaa pintaa syvemmältä. (Hyperledger 2022i.)



Kuva 7. Kuvankaappaus Logspout -työkalun tuottamasta tietovirrasta.

## 3.7 Ketjukoodin implementointi verkkoon

Tässä osiossa on tarkoituksena tehdä *Fabric-Samples*-paketin mukana toimitetun *Transfer-Assets-Basic*-ketjukoodin pohjalta uusi ketjukoodi. Lisäksi samaa pohjaa hyödynnetään varsinaisen toteutuksen pohjalla luvussa 4. Tarkoituksena on ennen kaikkea havainnollistaa koko prosessi uuden ketjukoodin luomiseksi ja sen käyttöönottamiseksi verkossa luvussa 3.5 kuvailtua tarkemmalla tasolla, jotta voitaisiin tutkia Fabricin keskeisiä toimintaperiaatteita ketjukoodien näkökulmasta vielä yksityiskohtaisemmin. Tässä osiossa ei kuitenkaan käydä läpi verkon pystyttämistä ja ketjukoodin implementointia enää komentorivitasolla. Tässä kuvattu ketjukoodi voidaan implementoida edelleen testiverkkoon, käyttäen hyväksi sen sisältämiä komentosarjoja. Vaihtoehtoisesti voidaan hyödyntää alkuperäisen Fabric-dokumentaation *Tutorials*-osiosssa, luvussa *Deploying a smart contract to a channel* esitettyjä komentoja, yhdistettynä tässä kuvattuihin, prosessin yksittäisiin vaiheisiin. (Hyperledger 2022a; Hyperledger 2021e.)

Aloitetaan uuden ketjukoodin kehittäminen kopioimalla *transfer-assets-basic*-kansiosta *chaincode-javascript*-kansion sisältö kokonaisuudessaan uuteen sijaintiin. Nimetään uusi kansio esim. muotoon ”omacc”, jotta tunnistetaan se myöhemmin. Avataan lisäksi *package.json*-tiedosto tekstieditoriin, muutetaan *name*-kentän arvoksi myös ”omacc” ja tallennetaan tiedosto. Tähän voi halutessaan täydentää myös esim. uuden versionumeron, ohjelman kuvauksen sekä tekijän ja lisenssin tiedot.

Seuraavaksi avataan *lib*-kansiossa oleva *assetTransfer.js*-tiedosto haluamaamme IDE-editoriin. Tiedoston nimen voi halutessaan myös muuttaa, mutta tällöin on muistettava muuttaa lisäksi juurikansiossa sijaitsevan *index.js* -tiedoston rivit vastaavasti:

```
const assetTransfer = require('./lib/assetTransfer');
module.exports.AssetTransfer = assetTransfer;
module.exports.contracts = [assetTransfer];
```

Avatusta ketjukoodista muokataan sen sisältämää *AssetTransfer* -luokan sisältöä, joka täydentää pääluokkaa *Contract.* Kaikki muut luokan sisältämät funktiot voitaisiin periaatteessa poistaa, lukuun ottamatta *InitLedger* -funktiota. Kuitenkin myös tämän sisältämät arvot (*assets*) voidaan nyt tyhjentää tai korvata uusilla. Saattaisi olla lisäksi hyödyllistä säilyttää joitakin valmiita funktioita, joista voi ottaa mallia omien funktioiden kirjoittamiseen.

Muodostetaan *assets*-muuttujaan seuraavanlaiset tietueet, jotka sisältävät yksittäisten resurssien arvot:

```
ID: 'asset1',
OmistajanNimi: 'Henri Tikkanen',
OmistajanId: '123456',
OmaisuudenId: '8489999447',
MerkintaAika: '01.01.2022',
OmaisuudenArvo: 99999,
```

Arvot voivat olla mitä tahansa sopivia merkkijonoja tai numeroarvoja ja tietueita voi monistaa rajattomasti. Testaustarkoitukseen riittää kuitenkin hyvin esim. 35 tietuetta, joilla on vähintään uniikki ID-tunniste. (Hyperledger 2021e.)

Tässä esimerkissä on tarkoitus toteuttaa funktio, joka palauttaa kaikki tietylle omistajalle rekisteröidyt resurssit, parametrina annetun *OmistajanId* -arvon perusteella. Mikäli yhtään hakua vastaavaa tulosta ei löydy, palautetaan ilmoitus. Lisäksi tarkistetaan annetun parametrin kelvollisuus, joka voi tässä tapauksessa muodostua vain kokonaisluvuista, ilman erikoismerkkejä tai välilyöntejä. Virheelliselle arvolle palautetaan virheilmoitus.

Pohjalla voidaan käyttää *GetAllAssets*-funktiota, joka palauttaa kaikki arvot ilman minkäänlaista suodatusta. Kyseessä on asynkroninen (*async*) funktio, joka käyttää tulosten iteroinnissa Fabricin sisäistä *getStateByRange*-metodia ilman hakua rajoittavia parametreja kaikkien pääkirjan sisältämien avain-arvo-parien poimimiseksi, jotka käsitellään yksitellen *while*-rakenteen sisällä. JSON-muotoiset objektit parsitaan *UTF8*-muotoisiksi merkkijonoiksi ja puskuroidaan binäärimuotoon sujuvan käsittelyn varmentamiseksi, käyttäen Node.js:n *buffer*-metodia. Uuden tuloksen käsittely alkaa vasta, kun edellinen on saatu varmuudella suoritettua. Kaikki ketjukoodin funktiot saavat *InitLedger*-funktion alustaman datasyötteen *ctx*-parametrin (lyh. termistä *custom context*) kautta. Lopuksi käsitelty tietomassa palautetaan JSON-merkkijonona, mikäli operaatio ei ole katkennut sitä ennen virheeseen. (Hyperledger 2021e.)

Nimetään uusi funktio kuvaavasti *GetAssetsByOwner*. Edellä kuvatun funktion perusrakenteeseen tarvitaan lisäksi suodatusarvon validointi, joka tehdään käyttäen säännöllistä lauseketta (engl. *regular expression*) ja JavaScriptin sisäistä *test*-metodia. Lisäksi tarvitaan luonnollisesti itse suodatus, joka toteutetaan yksinkertaisella *if*-lauserakenteella. Ehdon täyttävät tulokset lisätään tulokseen sisältävään *result*-taulukkoon. Mikäli taulukko on lopuksi edelleen tyhjä, palautetaan vain merkkijono: ”Ei yhtään tulosta tällä suodatuksella!”.

```
// Palauta kaikki resurssit, joiden omistajalla on annettu id
async GetAssetsByOwner(ctx, id) {
    // Arvon validointi, hyväksytään vain kokonaisluvut
    const validation = function(id) {
        let pattern = /^[0-9]+$/;
        let valid = pattern.test(id);
        return valid;
    }
    // Annetaan virhe ja keskeytetään pyyntö (status 500), mikäli arvo ei ole validi
    if (!validation(id)) {
        throw new Error(`Annettu id ${id} ei kelpaa!`);
    }
    // Alustetaan muuttujat ja käydään läpi kaikki resurssit
    let allResults = [];
    const iterator = await ctx.stub.getStateByRange('', '');
    let result = await iterator.next();
    while (!result.done) {
        const strValue = Buffer.from(result.value.value.toString()).toString('utf8');
        let record;
        try {
            record = JSON.parse(strValue);
        } catch (err) {
            console.log(err);
            record = strValue;
        }
        // Jos annettu arvo täyttyy, lisätään tuloksiin
        if (record.OmistajanId === id) {
            allResults.push(record);
        }
        result = await iterator.next();
    }
    // Palautetaan viesti jos tulos on tyhjä, muuten JSON-objektien arvot merkkijonoina
    if (!allResults.length) {
        return "Ei yhtään tulosta tällä suodatuksella!";
    }
    return JSON.stringify(allResults);
}
```

Kuva 8. Kuvankaappaus valmiista *GetAssetsByOwner*-funktiota

Kun uuden ketjukoodin käyttämät tiedostot on tallennettu, voidaan siirtyä riippuvuuksien asentamiseen tai päivittämiseen, verkon käynnistämiseen ja

ketjukoodin pakkaamiseen. Tämä kaikki voitaisiin suorittaa luvussa 3.4 kuvatulla tavalla, käyttäen Fabric-testipaketin mukana toimitettuja komentosarjoja, jolloin prosessi olisi suoraviivaisempi. Seuraavassa käydään läpi kuitenkin ne muutamat välivaiheet koko prosessin selkiyttämiseksi, jotka aiemmin suoritettiin valmiiden komentosarjojen avulla. (Hyperledger 2021e.)

Mikäli ketjukoodin rakentaminen olisi aloitettu tyhjästä, tulisi tarvittavat riippuvuudet asentaa yksitellen *npm i* -komennolla. Tässä kuitenkin oletetaan, että sisältö on kopioitu *asset-tranfer-basic*-paketista, joten riippuvuuksien tulisi olla valmiiksi listattuna ja asennettuna. Tarkistetaan kuitenkin seuraavien riippuvuuksien olevan listattuna *package.json*-tiedoston kohdassa *dependencies*:

1. *fabric-contract-api*
2. *fabric-shim*
3. *json-stringify-deterministic*
4. *sort-keys-recursive.*

Mikäli kaikki tarvittavat riippuvuudet ovat listattuna, voidaan asentaa uusimmat versiot tarvittavista NPM-paketeista *npm install* -komennolla ketjukoodin juurikansiossa. Tämän jälkeen myös *package-lock.json*-tiedosto samassa kansiossa olisi pitänyt päivittyä ja siinä tulisi nyt näkyä ketjukoodin uusi nimi. Tätä tiedostoa ei koskaan pidä itse muokata, vaan NPM-pakettienhallinta luo sen. Verkon käynnistäminen tapahtuu luvussa 3.4 kuvatulla tavalla. Lisäksi on suositeltavaa käynnistää samalla *Logspout*-työkalu logitietojen reaaliaikaisen tarkasteluun. (Hyperledger 2022a.)

Viimeistään tässä vaiheessa on määriteltävä *peer*-binääritiedostojen sijainti, jotta kaikki tarvittavat komponentit voidaan pakata mukaan ketjukoodin kanssa. Lisäksi asetetaan *FABRIC_CFG_PATH*-muuttuja osoittamaan *core.yaml*-määrittelytiedostoon. Nämä suoritetaan edelleen luvussa 3.4 kuvatulla tavalla ja tämä vaihe voitaisiin suorittaa myös ennen verkon käynnistämistä. On lisäksi syytä varmistaa, että *peer CLI* on käytössä, sillä tästä eteenpäin luvun kaikissa komennoissa oletetaan käytettävän *peer*-syntaksia ja sen *lifecycle*-alakomentoa. Syntaksin rakenne on tässä yhteydessä aina *peer lifecycle chaincode <alakomento>* -muodossa. Nämä komennot on tarkoitettu pääkäyttäjien vuorovaikutukseen vertaisten kanssa, käyttäen elinkaaritoimintoja. (Hyperledger 2022a.)

Mikäli *peer* on toiminnassa, voidaan siirtyä ketjukoodin pakkaukseen ja asentamiseen. Pakkaus suoritetaan käyttäen *package*-alakomentoa. Komennossa *path*-atribuutin jälkeen on pakattavan ketjukoodin sijainti ja *label*-atribuutin jälkeen sen tunniste, jossa myös versionumero. Versionumeron päivittäminen myös nimeen on selkeyden vuoksi erittäin suositeltavaa. Ennen asennusta suoritetaan vielä ympäristömuuttujien asetukset *Org1*-vertaisen osalta, joka toimii oletusportissa 7051, kuten luvussa 3.4 on kuvattu. Tämän jälkeen voidaan suorittaa ketjukoodin asennus vertaisorganisaatiolle tehdystä paketista käyttäen *install*-alakomentoa. (Hyperledger 2022a.)

Onnistuneen asennuksen jälkeen vastauksena pitäisi olla paluukoodi *200*, sekä paketin tunniste (engl. *package identifier*), joka koostuu pakkausvaiheessa annetusta tunnisteesta ja 64-merkkiä pitkästä, yksilöllisestä kirjain- ja numeroyhdistelmästä. Nyt voidaan suorittaa samat toimenpiteet *Org2*-vertaiselle. Tarvittavat ympäristömuuttujat ovat muilta osin samat, kuin *Org1:n* kohdalla, mutta CORE_PEER_TLS_ENABLED-muuttujaa ei tarvitse enää määritellä, kaikki *Org1*-viittaukset korvataan *Org2*:lla ja portiksi asetetaan oletuksena 9051. Lopuksi suoritetaan paketin asennus täysin samoin kuin edellä. (Hyperledger 2022a.)

Kun ketjukoodi on asennettu molemmille vertaisorganisaatioille, siirrytään sen määritelmän varmentamiseen kaikkien vertaisorganisaatioiden toimesta. Ensin on syytä varmistaa, että ketjukoodi on asennettu onnistuneesti *queryinstalled* -alakomennolla. Vastauksena tulisi saada edellä mainittu paketin yksilöllinen tunniste, joka tulee viedä muuttujan *CC_PACKAGE_ID* arvoksi. Tämän jälkeen voidaan suorittaa ketjukoodin hyväksyminen *Org2*-vertaisorganisaatiolle, jolle ympäristömuuttujat on viimeksi asetettu. Hyväksyminen tapahtuu käyttäen *approveformyorg*-alakomentoa, jossa tulee tarvittaessa muuttaa myös ketjukoodin nimi ja versio. Hyväksyntä on suoritettava kaikille vertaisorganisaatioille, joten asennetaan vielä uudestaan edellä mainitut ympäristömuuttujat kappaleen 3.4 mukaan *Org1*:lle, alkaen muuttujasta *CORE_PEER_LOCALMSPID*, jonka jälkeen suoritetaan edellä kuvattu ketjukoodin hyväksyntä *Org1*:lle täysin vastaavasti. (Hyperledger 2022a.)

Kun kaikki vertaisorganisaatiot ovat hyväksyneet ketjukoodin kuvauksen, voidaan se edelleen hyväksyä siirrettäväksi kanavalle. Hyväksyntöjen tilan voi

ensin tarkistaa käyttäen alakomentoa *checkcommitreadiness*. Organisaatioiden hyväksyttyä ketjukoodin kuvauksen, tulisi tästä muodostua JSON-muotoinen sanoma, jossa molempien vertaisorganisaatioiden kohdalla on *Approvals* -tietueen arvoina *true*. Mikäli molemmat vertaisorganisaatiot ovat hyväksyneet ketjukoodin kuvauksen, varsinainen kanavalle hyväksyntä voidaan suorittaa molemmille vertaisorganisaatioille käyttäen *commit*-alakomentoa ja sen *-o* -atribuuttia. On myös syytä tarkentaa, että vaikka suomen kielellä kahdessa edellisessä luvussa puhutaankin hyväksymisestä, kyseessä on kaksi erillistä vaihetta; ketjukoodin määritelmän hyväksyminen (engl. *approve*) sekä ketjukoodin hyväksymisestä verkkoon (engl. *commit*), joka voitaisiin myös kääntää esim. muotoon ”tekeminen”. (Hyperledger 2022a.)

Hyväksymiskomennon jälkeen odotetaan, kunnes kaikki vertaisorganisaatiot ovat hyväksyneet siirron. Tarvittaessa hyväksyntöjen tilan voi tarkistaa *query-committed*-alakomennolla. Kun ketjukoodi on hyväksytty ja otettu käyttöön kanavalla, voidaan siirtyä alustamaan ketjukoodin sisältämät alkuarvot pääkirjalle käyttäen *invoke*-alakomentoa ja sen *-o* -atribuuttia. Nyt ketjukoodi tulisi olla käytettävissä samoin kuin testiverkossakin ja voitaisiin esimerkiksi kokeilla resurssien hakua omistajan tunnisteella, käyttäen edellä luotua *GetAssetsByOwner*-funktiota ja haluttua arvoa *peer chaincode query* -komennon argumentteina. (Hyperledger 2022a.)

# 4 ÄLYKKÄÄN SOPIMUKSEN TOTEUTUS

Tässä luvussa käydään läpi prosessi uuden älykkään sopimuksen kehittämiseksi ja sen implementoimiseksi Hyperledger Fabric -verkkoon. Tarkoituksena on tutkia tyypillistä prosessia yksinkertaisen esimerkin kautta, joka kuitenkin voisi vastata johonkin tosielämän käyttötarkoitukseen. Lähtökohtana käytetään luvussa 3.8 muokattua versiota *Asset Transfer Basic* -ketjukoodista ja siihen toteutettua funktiota omistusten suodattamiseen henkilötunnisteella. Tässä kuitenkin toteutetaan kokonaan uusi ketjukoodi, jonka olisi tarkoitus vastata luvussa 4.1 kuvattua taustoitusta sekä siihen perustuvaa vaatimusten määrittelyä luvussa 4.2.

## 4.1 Toteutuksen tausta

Tässä on kyseessä eräs keinotekoinen esimerkki, jonka kuitenkin olisi tarkoitus vastata realistista skenaariota tosielämän mahdollisesta toimijasta, jolla on muuttuvasta toimintaympäristöstä nouseva selkeä tarve järjestelmäuudistukselle. Kuvitteellisena referenssinä toteutukselle on suomalainen finanssialan toimija, joka ylläpitää rekisteriä asiakkaidensa omistamista yksittäisistä omaisuuseristä, luo niistä tilastoja ja tarjoaa tietoja loppuasiakkailleen verkkopalvelun välityksellä. Ongelmana on ollut mm. nykyisen järjestelmän raskas ylläpito, jatkuvasti lisääntyvät tietoturvakustannukset, viranomaisten koventuneet vaatimukset henkilötietojen käsittelyssä, huoli järjestelmän skaalautuvuudesta tieto- ja käyttäjämäärän voimakkaasti kasvaessa sekä halutun tiedon hallittavuus eri rekistereistä.

Tällä hetkellä suurin osa kaikesta tiedosta sijaitsee keskitetysti ulkopuolisen palveluntarjoajan hallinnoimilla palvelimilla Suomessa. Erityisesti toimijan Pohjois-Amerikkalainen emoyhtiö on ilmaissut huolensa palvelinten sijainnista itäisen Euroopan epävakaalta vaikuttavassa geopoliittisessa ympäristössä ja lisääntyneistä kyberhyökkäyksistä läntisiä finanssilaitoksia kohtaan. Myös tiedon siirtämistä kauemmaksi varsinaisista käyttäjistä pidetään kuitenkin huonona ratkaisuna, sillä tämä voisi mm. hankaloittaa ongelmiin reagoimista paikallisella tasolla sekä heikentää tietoa käyttävien järjestelmien suorituskykyä loppukäyttäjien näkökulmasta. Lisäksi tulee ottaa huomioon mm. EU:n yleisen tietosuojasetuksen (*GDPR*) rajoitukset henkilötietoja sisältävän datan säilyttämisestä EU:n alueen ulkopuolella (Euroopan parlamentin ja neuvoston asetus (EU) 2016/679, 5. luku)

Näin ollen on lähdetty etsimään vaihtoehtoja tiedon keskittyneelle hallinnoinnille perinteisissä tietokannoissa. Oletuksena olisi, että alkuperäisen datan hallinnointi yksityisessä lohkoketjussa, erityisesti laajempien tietomassojen ja sensitiivisen tiedon kyseessä ollen, toisi etuja parantuneen turvallisuuden ja joustavuuden kautta verrattuna nykyiseen malliin. Uudella ratkaisulla odotetaan saavutettavan pidemmällä aikavälillä myös merkittäviä kustannussäästöjä verrattuna raskaisiin suojauksiin, joita finanssialan tietojärjestelmät perinteisesti vaativat.

## 4.2 Vaatimusten määrittely ja suunnittelu

Toimeksianto on kerätä tilastotietoja lohkoketjuverkosta organisaation sisäiseen käyttöön sekä valikoituja tietoja jatkokäsittelyä varten. Tietoturvan ja tietojärjestelmän joustavuuden kehittäminen kaikilla osa-alueilla ovat järjestelmäuudistuksen keskeisiä tavoitteita.

Tilastointia varten yksittäiset tiedot halutaan anonymisoida niin, ettei yksittäisten resurssien omistajia ole mahdollista suoraan tunnistaa suodatetusta tietomassasta, täyttäen myös GDPR-asetusten vaatimukset. Tiedot tulostetaan toteutuksen tässä vaiheessa vain ruudulle, mutta myöhemmässä vaiheessa voisi olla mahdollista myös esimerkiksi viedä ne erilliseen tiedostoon jatkokäsittelyä varten. Lisäksi halutaan mahdollisuus viedä tietoja rajapinnan avulla ulkoiseen järjestelmään, kuten toimijan omaan verkkopalveluun, jossa resurssien omistajat voivat itse tarkastella niitä vahvan tunnistautumisen kautta. Tätä ei kuitenkaan toteuteta vielä ensimmäisessä vaiheessa.

Jotta voitaisiin vähimmäistasolla tilastoida hallinnassa olevia omaisuuseriä sekä yhdistää tietyt omaisuuserät tietyille asiakkaille, omaisuuseriä voisi olla tarkoituksenmukaista suodattaa ainakin arvon ala- ja ylärajojen perusteella sekä henkilötunnisteella.

Seuraavat tiedot halutaan suodattaa omaisuuserien arvojen perusteella anonyymisti tilastointia varten:

1. hakuun täsmäävien arvojen kokonaismäärä
2. hakuun täsmäävien arvojen keskiarvo
3. hakuun täsmäävien arvojen mediaani
4. hakuun täsmäävien arvojen kokonaisarvo.

Seuraavat tiedot halutaan suodattaa asiakkaan henkilötunnisteen perusteella ja viedä ulkoiseen järjestelmään:

1. omaisuuserän omistajan nimi
2. omaisuuserän omistajan tunniste
3. omaisuuserän tunniste
4. omaisuuserän merkintäaika
5. omaisuuserän arvo.

Erityyppisten kyselyiden yhdistäminen haluttiin estää, jotta yksittäisiä tilastotietoja ei voitaisi liittää niiden omistajiin. Suodatusarvot toteutuksen

ensimmäiseen vaiheeseen valittiin sen perusteella, että ne kuvastaisivat joitakin finanssialan toimijoiden käyttämiä, tyypillisiä perustietoja yksittäisille omaisuuserille sekä tilastoinnissa yleisesti käytettyjä hakuehtoja. Myös uusien hakuehtojen lisäys myöhemmässä vaiheessa olisi verrattain helppoa.

### 4.3 Toteutus

Tässä luvussa raportoidaan mallitoteutus ketjukoodista, joka pyrkii täyttämään edellä määritellyt tiedon hallintaan ja käyttöön liittyvät vaatimukset, tämän opinnäytetyön kontekstissa tarkoituksenmukaisessa laajuudessaan. Tässä toteutuksessa ei siis painoteta esimerkiksi ketjukoodin käytettävyyttä loppukäyttäjien näkökulmasta tai oteta millään tavoin kantaa käyttäjäorganisaation sisäisiin prosesseihin ketjukoodin käyttöönottamiseksi omassa toiminnassaan. Toteutuksen ensisijaisena tarkoituksena on tutkia ja havainnoida ketjukoodin toteutukseen liittyvää prosessia Hyperledger Fabricin näkökulmasta sekä edelleen tarkentaa siihen liittyviä yksityiskohtaisempia vaatimuksia ja mahdollisia haasteita edellisten kappaleiden perusteella. Valmiin ohjelman koodi esitetään jäljempänä kuvankaappausten muodossa, pilkottuna neljään loogiseen osa-alueeseen. Koodin kaikki keskeiset elementit on myös kommentoitu suomeksi helpottamaan hahmottamista suhteessa raporttiin.

Toteutuksen pohjaksi otettiin luvussa 3.7 toteutetun ketjukoodin sisältö, sisältäen valmiiksi ketjukoodin olennaiset rakenteelliset elementit, perustoimintoihin käytettävät funktiot sekä valmiin funktion resurssien suodatukseen henkilönumeron perusteella. Tiedostosta *AssetsTranfer.js* tarkistettiin, että tiedoston alkuun oli liitetty paketit *fabric-contract-api*, *sort-keys-recursive* ja *json-stringify-deterministic*. Seuraavaksi muutettiin luokan nimeksi *FilterAssets* ja varmistettiin, että se jatkaa (*extends*) luokkaa *Contract.* Tiedosto tallennettiin nimellä *filterAssets.js lib*-hakemistoon sekä muutettiin samalla ketjukoodin juurihakemistossa olevan *index.js*-tiedoston kaikki luokkaviittaukset osoittamaan juuri luotuun *FilterAssets*-luokkaan.

Ohjelmaan otettiin pakollisen *InitLedger*-funktion lisäksi Fabric-malleissa esitellyt funktiot *CreateAssets*, *UpdateAsset*, *DeleteAsset* ja *TransferAssets*, joista kaikkien ajateltiin vastaavan johonkin realistiseen käyttötapaukseen vaatimusmäärittelyn perusteella. Edellä mainittujen funktioiden nimet ilmentävät

niiden käyttötarkoituksia erittäin hyvin, jonka tässä yhteydessä on ajateltu olevan riittävä tieto. Alkuperäisiä funktioita ei myöskään ollut tarpeen muokata muilta osin, kuin muutettujen ominaisuuksien osalta. Siis esimerkiksi kaikista funktioista, joissa käytettiin alkuperäistä ominaisuutta nimeltään *Owner*, tuli tämä muuttaa muotoon *OmistajanNimi*. Nimeämiseen käytettiin ns. *CamelCase*-muotoilua.

Toimiva funktio omaisuuserän suodatukseen asiakkaan henkilötunnisteen perusteella oli jo valmiina (*GetAssetsByOwner*), jonka lisäksi tarvittiin vielä suodatusfunktion toteutus. Tämän toteutukseen voitiin myös hyödyntää aiemman -funktion perusrakennetta, jota laajennettiin tarpeen mukaan. Ketjukoodin testaukseen käytettiin myös samoja resurssien alkuarvoja. Tuotantokäyttöön tarkoitettua sovellusversiota toteutettaessa alkuarvot jätettäisiin kuitenkin todennäköisesti tyhjiksi.

Uuden funktion nimeksi asetettiin kuvaavasti *GetFilteredAssets* ja sen rakenne voitiin jakaa seuraaviin loogisiin osa-alueisiin:

1. syötteen validointi ja yleinen virheenhallinta
2. resurssien iteratiivinen prosessointi ja suodatus
3. laskuoperaatiot
4. tietojen tulostus.

Validoinnin toteutukseen tarvittiin nyt kahden arvon tarkistus ja myös desimaaliluvut oli hyväksyttävä. Tämän vuoksi arvojen tarkistus toteutettiin *if*-lausekkeella, joka palauttaa arvon *true*, mikäli molemmat arvot hyväksytään. Muilta osin toteutus oli aiemman kaltainen, kuitenkin sillä erotuksella, että käytettiin hieman tarkennettua säännöllistä lauseketta.

```
166     // Palauta kaikki resurssit, joiden arvo on annettujen arvojen välillä
167     async GetFilteredAssets(ctx, value1, value2) {
168         // Arvon validointi, hyväksytään vain kokonaisluvut ja desimaaliarvot
169         const validation = function(value1, value2) {
170             let pattern = /^-?\d+\.?\d*$/;
171             let valid = false;
172             if (pattern.test(value1) && pattern.test(value2)) {
173                 valid = true;
174             }
175             return valid;
176         }
177         // Annetaan virhe ja keskeytetään pyyntö (status 500), mikäli arvo ei ole validi
178         if (!validation(value1, value2)) {
179             throw new Error(`Annetut arvot ${value1} ja ${value2} eivät kelpaa!`);
180         }
```

Kuva 9. Kuvankaappaus valmiin *GetFilteredAssets*-funktion ensimmäisestä osa-alueesta, jossa näkyy syötteen validointi ja yleinen virheenhallinta

Ennen iteroinnin aloitusta alustettiin muuttujat *allResults* (taulukko, kaikki tulostettavat arvot), *i* (kokonaismäärä), *sum* (summa), *median* (taulukko, mediaani) ja *avg* (keskiarvo). Nämä muuttujat alustettiin muodossa *let*, sillä niiden arvoja oli tarvetta muuttaa suorituksen aikana. Myös varsinainen arvojen iterointi toteutettiin vastaavasti, kuin aiemmassakin toteutuksessa käyttäen Fabricin *getStateByRange*-metodia. Iteraatioprosessin sisälle kuitenkin lisättiin tarkistus molempien annettujen filtteriarvojen perusteella niin, että mukaan otetaan tulokset, jotka ovat yhtä suuret tai suuremmat kuin ensimmäinen arvo ja yhtä suuret tai pienemmät kuin toinen arvo. Parsittu arvo saatiin *record*-muuttujasta, avaimella *OmaisuudenArvo*. Lopuksi päivitettiin hyväksyttyjen arvojen kokonaismäärä ja arvojen yhteissumma sekä lisättiin arvo mediaanin laskemisessa käytettävään taulukkoon.

Arvojen iteroinnin jälkeen lisättiin tarkistus, sisältääkö taulukko *allResult* arvoja. Mikäli taulukko on tyhjä, palautetaan siitä ilmoittava viesti, muussa tapauksessa jatketaan mediaanin laskentaan. Mediaani lasketaan ensin etsimällä kaikki arvot sisältävän taulukon keskimmäinen indeksi, järjestelemällä taulukon arvot numerojärjestykseen pienimmästä suurimpaan ja lopuksi palauttamalla kyseinen keskimmäinen arvo, mikäli arvojen kokonaismäärä ei ole parillinen, muuten keskimmäisten kahden luvun keskiarvo. Lisäksi lasketaan kaikkien arvojen keskiarvo summasta sekä kokonaismäärästä ja kaikki lasketut tilastot koostetaan lopuksi *stats*-muuttujaan.

```
181         // Alustetaan muuttujat
182         let allResults = [];
183         let i = 0;
184         let sum = 0;
185         let median = [];
186         let avg = 0;
187         // Käydään läpi kaikki resurssit
188         const iterator = await ctx.stub.getStateByRange('', '');
189         let result = await iterator.next();
190         while (!result.done) {
191             const strValue = Buffer.from(result.value.value.toString()).toString('utf8');
192             let record;
193             try {
194                 record = JSON.parse(strValue);
195             } catch (err) {
196                 console.log(err);
197                 record = strValue;
198             }
199             // Jos annettu arvo täyttyy, lisätään tuloksiin
200             if (record.OmaisuudenArvo >= value1 && record.OmaisuudenArvo <= value2) {
201                 // Otetaan talteen vain arvo ja id
202                 const anonRecord = {
203                     ID: record.ID,
204                     OmaisuudenArvo: record.OmaisuudenArvo,
205                 };
206                 allResults.push(anonRecord);
207                 // Lisätään yksittäiset arvot muuttujiin
208                 i++;
209                 sum += record.OmaisuudenArvo;
210                 median.push(record.OmaisuudenArvo);
211             }
212             result = await iterator.next();
213         }
214         // Palautetaan viesti jos tulos on tyhjä, muuten JSON-objektien arvot merkkijonoina
215         if (!allResults.length) {
216             return "Ei yhtään tulosta tällä suodatuksella!";
217         }
```

Kuva 10. Kuvankaappaus valmiin *GetFilteredAssets*-funktion toisesta osa-alueesta, jossa näkyy muuttujien alustus sekä resurssien iteratiivinen prosessointi ja suodatus

```
218         // Lasketaan mediaani ja keskiarvo
219         const mid = Math.floor(median.length / 2),
220         nums = [...median].sort((a, b) => a - b);
221         const med = median.length % 2 !== 0 ? nums[mid] : (nums[mid - 1] + nums[mid]) / 2;
222         avg = sum / i;
```

Kuva 11. Kuvankaappaus valmiin *GetFilteredAssets*-funktion kolmannesta osa-alueesta, jossa näkyy laskuoperaatiot

Funktion toteutuksen viimeinen vaihe oli kaikkien *allResults*-taulukkoon kerättyjen arvojen muuntaminen merkkijonoiksi ja palautus yhdessä tilaston kanssa, joka tulostetaan kaikkien arvojen perään selkeästi rivitettynä. Lopuksi tarkistettiin vielä, että luokan sulkemisen jälkeen se viedään *module.exports*-metodilla, nimellä *FilterAssets*.

```
223         // Koostetaan tilasto
224         let stats = `
225             Osumia yhteensä: ${i}
226             Osumien keskiarvo: ${avg}
227             Osumien mediaani: ${med}
228             Osumien kokonaisarvo: ${sum}`;
229
230         // Palautetaan suodatetut tulokset ja tilasto
231         return JSON.stringify(allResults) + "\n" + stats + "\n";
232     }
```

Kuva 12. Kuvankaappaus valmiin *GetFilteredAssets*-funktion neljännestä osa-alueesta, jossa näkyy tietojen tulostus

Lisäksi huomioitiin, että alkuperäinen toteutuksen pohjalle kopioitu ohjelma oli julkaistu avoimen koodin *Apache 2.0* -lisenssin alla. Tässä tapauksessa Fabric-mallien alkuperäiset oikeudet omistaa IBM, mutta *Apache 2.0* -lisenssi sallii lähdekoodin vapaan käytön ja kehittämisen joko avoimeen tai suljettuun tarkoitukseen, edellyttäen kuitenkin alkuperäisen tekijänoikeuden huomautuksen sisällyttämisen myös muutettuun versioon (Apache 2.0, 2022).

Lopuksi valmiin ketjukoodin kaikki riippuvuudet päivitettiin *npm install* -komennolla ja ketjukoodi pakattiin komennolla *peer lifecycle chaincode package*, nimellä *filter-assets_1.0*. Nyt ketjukoodi oli valmis otettavaksi käyttöön verkossa testausta varten.

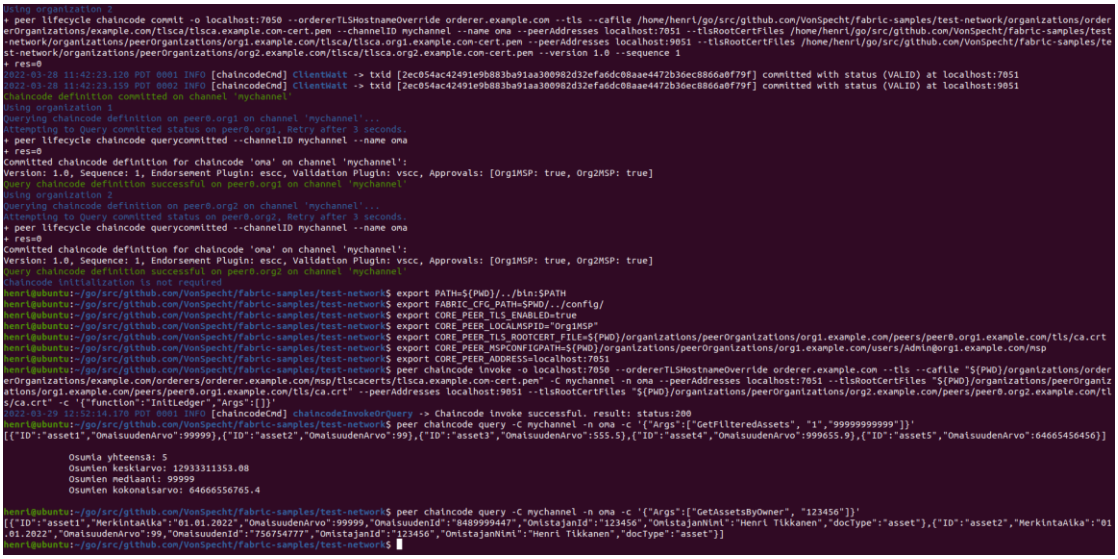
### 4.4 Testaus ja yhteenveto

Kun ketjukoodi oli hyväksytty verkkoon molempien vertaisorganisaatioiden toimesta, voitiin aloittaa sen testaus. Testaukseen käytettiin valmiiksi ketjukoodille määriteltyjä alkuarvoja. Yhtä hyvin olisi voitu syöttää tässä vaiheessa uusiakin arvoja ketjukoodin sisältämän *CreateAssets*-funktion avulla. Testauksessa syötettiin mahdollisimman erilaisia arvoja kahdelle itse toteutetulle funktiolle, jotta voitiin varmistua, että ne toimivat kaikenlaisilla hyväksytyillä arvoilla tarkoitetulla tavalla ja toisaalta palauttivat oikeanlaisen virheen, mikäli syötetty arvo ei ollut hyväksytty.

Varsinainen testaus suoritettiin tässä tapauksessa yksinkertaisena käyttäjätestauksena, eikä mitään automaatiotestausta ollut tarkoituksenmukaista suorittaa. On kuitenkin syytä mainita, että Fabric-mallit sisältävät myös Transfer Assets Basic -ketjukoodille testauspaketin automaatiotestausta varten, jonka

voisi halutessaan kopioida ja muokata tarpeen mukaan. Muokattava JavaScript-tiedosto löytyy alkuperäisen ketjukoodin alakansiosta *test*.

Kun ketjukoodin testauksessa havaitaan virhe toteutuksessa, tulee lähtökohtaisesti verkko ajaa alas, tehdä korjaukset ketjukoodin, paketoida se uudelleen, käynnistää verkko uudelleen ja käydä läpi kaikki elinkaaritoiminnot uudelleen muutetun ketjukoodin käyttöönottamiseksi verkossa. Tässä suhteessa testausprosessi Hyperledger Fabric -verkossa eroaa eniten tyypillisestä ohjelmistokehitysprosessista, jossa muutetun ohjelmakoodin testaus kehitysympäristössä on yleensä erittäin suoraviivaista. Fabricissa olisi käytettävissä eräänlainen kehitystila (engl. *development mode*), jossa voidaan ohittaa elinkaaritoiminnot ketjukoodin päivityksen jälkeen. Myös tätä kokeiltiin käytännössä kehitysprosessin alkuvaiheessa, mutta testauksessa ei ilmennyt suoranaista tarvetta sen käytölle, sillä valmista koodia ei juurikaan tarvinnut tässä tapauksessa muokata. Kehitystila vaatii hieman poikkeavan prosessin ketjukoodin ottamiseksi käyttöön, jonka tarkka kuvaus löytyy Fabricin dokumentaatiosta (Hyperledger 2022g).




Kuva 13. Kuvankaappaus *GetFilteredAssets* ja *GetAssetsByOwner* -funktioiden testauksesta

Yhteenvetona voidaan todeta, että varsinainen toteutus oli melko suoraviivainen prosessi. Toteutetuista kahdesta funktiosta laajemmankin ohjelmakoodista tuli lopulta verrattain hyvin tiivis, käsittäen alle 100 riviä koodia kommentteineen. Selkein eroavaisuus verrattuna perinteiseen sovelluskehitysprosessiin ilmenikin testauksen yhteydessä. Itse ohjelmakoodin kehitykseen liittyen oli huomioitava olennaisten Fabric-moduulien liittäminen ohjelmaan, sekä

toteutetun uuden luokan periyttäminen *Contract*-pääluokasta. Myös alkuarvojen alustukseen käytettävän *InitLedger*-funktion oli oltava olemassa ja asetettava *ctx*-parametrin sisältö avain-arvo-pareina pääkirjaan, käyttäen *putState*-metodia, vaikka alkuarvot olisikin jätetty tyhjiksi. Vastaavasti tiedot haettiin pääkirjasta käyttäen iteroivaa *getStateByRange*-metodia ilman hakualueen määritystä. Ketjukoodin toteutuksessa välttämättömänä voidaan siis pitää myös tiettyjen Fabricin perusfundamenttien tuntemista, jotta osataan esimerkiksi käyttää oikeita metodeja pääkirjan tietojen käsittelyyn oikeassa järjestyksessä, mutta vastaavanlaisia asioita on yleensä huomioitava minkä tahansa sovelluskehitysprosessin yhteydessä, johon liittyy aiemmin toteutettuja rakenteita.

# 5 PÄÄTÄNTÖ

Tämän opinnäytetyön teoreettisessa osuudessa käytiin lyhyesti läpi älykkäiden sopimuksien historia, lohkoketjuteknologian syntyminen ja näiden kahden teknologian linkittyminen toisiinsa. Älykkäiden sopimusten historiaan liittyvästä materiaalin perusteella voidaan todeta, että teoreettinen perusta älykkäiden sopimusten tutkimukselle ja käytölle on luotu jo 90-luvulla, vaikka käsite useimmiten liitetäänkin vasta lohkoketjuteknologian myötä avautuneisiin uusiin mahdollisuuksiin. Voitiin lisäksi todeta, että lohkoketjuteknologian kehityksen myötä avautunut konkreettinen mahdollisuus älykkäiden sopimuksien taltioimiseksi hajautettuun tietorakenteeseen, täydensi Nick Szabon idean autonomisesti toimivista sopimuksista.

Tutustumalla lohkoketjuteknologiaan myös tietoturvan näkökulmasta, voitiin huomioida, että tietoturvan kolmesta peruskäsitteestä (ns. CIA-kolmio) tiedon eheys ja saatavuus ovat lohkoketjuteknologian ilmeisiä vahvuuksia verrattuna perinteisiin keskitettyihin tietorakenteisiin. Lisäksi teoriaosuudessa tutustuttiin Hyperledger Fabric -järjestelmän perusteisiin sekä hieman tarkemmin mm. salauksen ja älykkäiden sopimusten rooliin Fabricin perusarkkitehtuurissa. Kovin syvälle teknisiin konsepteihin ei kuitenkaan pureuduttu, sillä tämä opinnäytetyö pyrki lähestymään varsinaista aihettaan teoreettista tarkastelua painokkaammin empiiristen havaintojen kautta.

Yhtenä opinnäytetyön keskeisenä tutkimustavoitteena oli selvittää, millaiseen käyttöön Hyperledger Fabricin kaltaiset yksityiset lohkoketjut voisivat parhaiten soveltua ja mitä tulisi huomioida oikeantyyppisen lohkoketjuympäristön valinnassa. Tutustuin joihinkin erittäin vaativiin Hyperledger Fabricilla toteutettuihin tosielämän esimerkkeihin, joiden perusteella voidaan mielestäni todeta, että älykkäitä sopimuksia hyödyntämällä Fabric -lohkoketjuverkossa pystytään ratkaisemaan monimutkaisiakin ongelmia, joissa vaaditaan luotettavuutta, vahvaa yksityisyyttä ja ylipäätään korkeaa toteutuksen kokonaistasoa. On mielestäni perusteltua todeta, että esitellyt toteutukset ratkaisevat monipuolisesti erilaisia ongelmia, jotka koskettavat niin globaalia liike-elämää, viranomaistoimintaa kuin esimerkiksi suurten kuluttajaryhmien jokapäiväistä turvallisuuttakin. Voidaan myös olettaa, että useissa tapauksissa palveluiden loppukäyttäjät, kuten kuluttaja-asiakkaat, eivät ole laajasti tietoisia lohkoketjuteknologian roolista toteutuksen taustalla, sillä lohkoketjuteknologian läsnäolo ei yleensä mullista käyttäjäkokemusta yhtä näkyvällä tavalla, kuin esimerkiksi *HTTP*-protokolla teki aikoinaan.

Kuitenkin suhteuttamalla esimerkiksi tunnettujen toteutusten määrää maailmanlaajuisesti vastaavantyyppisiin, perinteiselle arkkitehtuurille pohjautuvien järjestelmien jatkuvasti toteutettavaan määrään, voidaan todeta Hyperledger Fabriciin pohjautuvien toteutuksien määrällisen kasvun olevan edelleen marginaalista. Perustuen luvussa 2.8 esitettyyn lähdemateriaaliin ja päätelmiin, voidaan pitää ilmeisenä, että yleisellä tasolla lohkoketjuteknologian yleistymistä rajoittavia tekijöitä ovat tällä hetkellä ainakin osaajien saatavuus ja tilaajien käytettävissä olevat muut resurssit. Varsinaisen lähdemateriaalin ulkopuolelta voidaan lisäksi olettaa, että rajallisesta lohkoketjuosaajien määrästä vain pieni murto-osa on erikoistunut Hyperledger Fabriciin.

Tyypillisiä Hyperledger Fabricilla toteutettuja tietojärjestelmiä edustavat aineiston perusteella esimerkiksi valtiollisten toimijoiden ja finanssisektorin korkeaa yksityisyyden tasoa ja tietoturvaa vaativat järjestelmät. Tämänkaltaiset toteutukset vaativat paitsi syvällistä osaamista Hyperledger Fabric -järjestelmästä, usein myös vuosien suunnittelua ja muuta huolellista valmistelua. Erityisen vaativia ovat yleisestikin migraatiot vanhoista järjestelmistä uusiin, kun ollaan tekemisissä sensitiivisen tiedon kanssa. Kuitenkin siirryttäessä täysin toisenlaiseen tapaan hallita ja varastoida tietoja esimerkiksi perinteisistä SQL-

tyyppisistä tietokannoista, migraation vaativuustaso saattaisi joissakin tapauksissa muodostua jopa ratkaisevaksi esteeksi lohkoketjupohjaiseen järjestelmään siirtymiselle. Migraatioihin liittyviä haasteita ei kuitenkaan tutkittu tässä opinnäytetyössä.

Voidaan myös todeta, että Hyperledger Fabric olisi todennäköisesti toteutukseltaan lian raskas ja erityisesti käynnistysvaiheessa liian laajoja resursseja vaativa sellaisiin käyttökohteisiin, joissa tarkoituksena on saada käyttöön lohkoketjuteknologian edut matalalla kynnyksellä, kuten erilaiset kuluttajille suunnatut palvelut asiakaspalvelun täydentämiseksi tai liiketoiminnan tehostamiseksi. Tällaisiin käyttökohteisiin jonkin valmiiksi toiminnassa ja laajassa käytössä olevan julkisen lohkoketjuverkon – kuten *Ethereumin* – hyödyntäminen olisi todennäköisesti parempi vaihtoehto. Hyperledger Fabricille perustuva lohkoketjuverkko ei yleensä olisi myöskään ensisijainen vaihtoehto sellaiseen käyttökohteeseen, jossa tiedon läpinäkyvyys ja laaja hajautuksen taso olisivat keskeisiä lähtökohtia.

Loppupäätelmänä erityyppisten lohkoketjuverkkojen eroavaisuuksista voitaisiinkin todeta, että Fabricin kaltaiset käyttöoikeuskontrolloidut lohkoketjuverkot voisivat todennäköisesti ratkaista joitain julkisiin lohkoketjuihin liitettyjä keskeisiä ongelmia, erityisesti liittyen yksityisyyteen, turvallisuuteen ja skaalautuvuuteen, mutta lopulta sopiva lohkoketjuverkko on valittava tiukasti käyttötarkoituksen perusteella. Älykkäiden sopimusten toteutus puolestaan poikkeaa eri lohkoketjuverkoissa lähinnä vain käytettävien ohjelmointikielien osalta ja käytännössä älykkäät sopimukset ovat vain verkkoon liitettyjä sovelluksia. Näin ollen niiden toteutustapaa ei nähdä yleensä käyttökohteita olennaisesti määrittävänä tekijänä.

Opinnäytetyön toteuttava osuus painottui kokeiluille valmiiden Fabric-mallien pohjalta, päätyen toimivan ketjukoodin toteutukseen. Keskeisenä lähdemateriaalina toimi Fabricin oma dokumentaatio, jonka havaitsin olevan pääasiassa erittäin hyvin laadittu ja ylläpidetty. Fabric-mallien avulla oli puolestaan mahdollista kokeilla verkon toimintaa matalalla kynnyksellä ja samalla oppia ymmärtämään järjestelmän toimintaperiaatteita paremmin. Fabric-mallien muokkaamiseen ja käyttöönottamiseen lohkoketjuverkossa riitti sen dokumentaatiossa kuvattujen esimerkkien huolellinen seuraaminen ja perustason ymmärrys

taustalla olevasta teknologiasta. Hyötyä oli luonnollisesti myös JavaScript-osaamisesta sekä perustason Linux-käyttökokemuksesta, näitä käytettäessä. Käytetyt kehitystyökalut olivat minulle pääosin ennestään tuttuja muusta sovelluskehityksestä, joten voidaan todeta, että älykkään sopimuksen kehitys Hyperledger Fabric -verkkoon ei asettanut mitään ratkaisevia erityisvaatimuksia käytettyjen kehitystyökalujen osalta.

Toisaalta kuitenkin havaitsin, että luotettavan tiedon saatavuus rajoittui usein Fabricin omiin dokumentteihin ja erityisesti syvemmälle Fabricin arkkitehtuuriin liittyvien dokumenttien ymmärtäminen vaati usein myös pelkästään Fabriciin liittyvän erikoistermistön hyvää hallintaa. Myös suomennoksien löytäminen joillekin termeille tuotti hankaluuksia, jotka eivät kuuluneet edes lohkoketjuteknologian yleiseen sanastoon, vaan niitä käytettiin kuvaamaan ainoastaan tiettyjä, erityisesti Fabriciin liittyviä prosesseja. Yksi esimerkki tällaisesta oli pääkirjan tekniseen rakenteeseen liittyvä englanninkielinen termi ”World State”, jolle valitsemani suomennos ”maailman tila” saattaisi luontevammin liittyä täysin toisenlaisiin aiheisiin. Ylipäätään luotettavan suomenkielisen lähdetiedon puuttuminen Fabricin teknisempiin konsepteihin liittyen oli ilmeistä. Myös saatavilla olevan englanninkielisen lähdetiedon koin kapea-alaiseksi, verrattuna esimerkiksi julkisen Ethereum-lohkoketjuverkon älykkäistä sopimuksista saatavilla olevaan tietoon. Fabricin oman dokumentaation ja tieteellisten artikkelien lisäksi tutustuin joihinkin järjestelmän kehitykseen liittyviin keskusteluihin Fabricin avoimilla Slack ja GitHub -kanavilla, joiden myötä aiheeseen liittyvä termistö tuli tutummaksi ja löysin vastauksia yksittäisiin teknisiin kysymyksiin.

Johtopäätöksenä tiedon saatavuudesta voisikin todeta, että Hyperledger Fabric ei ole laajalle yleisölle suunnattu järjestelmä – vaikka sillä voidaan ehkä ratkaista laajojakin yleisöjä koskettavia kysymyksiä – vaan verrattain marginaalinen aihe jopa lohkoketjuteknologian sisällä. Näin ollen on myös luonnollista, että saatavilla oleva tieto ei ole laajuudessaan samalla tasolla yleisempään käyttöön levinneiden järjestelmien kanssa.

Kun puhutaan minkä tahansa uuden teknologian käyttämisestä, on toki yleensä syytä kysyä ensin, ratkaiseeko se jonkin ongelman, jota jollain toisella teknologialla ei olisi kyetty ratkaisemaan. Tämä kysymys on aiheellinen – tai peräti keskeinen – myös tämän opinnäytetyön aiheen ja sille valitun

lähestymistavan näkökulmasta. Esimerkkitoteutus tarjosi yksinkertaisen ratkaisun melko tarkoin rajattuun ongelmaan, joka kuvattiin luvussa 4.1. On syytä huomata, että toteutettu ohjelma olisi voinut toimia lähes minkä tahansa modernin tietojärjestelmän kanssa. Tässä toteutuksessa oleellista ei kuitenkaan ollut itse ohjelma, vaan järjestelmä, joka käytti toteutettua ohjelmaa, eli älykästä sopimusta. Tämän opinnäytetyön tutkimusasetelman kannalta keskeisiä kysymyksiä siis olivat esimerkiksi, kuinka älykkään sopimuksen implementointi lohkoketjuun onnistui, miten tämä prosessi erosi siitä, että ohjelma olisi sen sijaan taltioitu esimerkiksi keskitettyyn tietokantaan ja mitä ongelmia lohkoketjun käyttäminen voisi ratkaista kuvatussa tapauksessa sen sijaan, että käytettäisiin tavanomaisempia keskitettyjä järjestelmiä.

Vastauksia ei mielestäni voida täysin kattavasti antaa tämän toteutuksen perusteella, sillä näin rajallisessa toteutuksessa ei ole mahdollista huomioida kaikkia toimivan järjestelmän kannalta merkityksellisiä tekijöitä, kuten suorituskykyä ja kustannuksia. Toteutuksessa ei myöskään voitu huomioida esimerkiksi rajapintojen kautta liitettyjä ulkoisia järjestelmiä ja niiden mukanaan mahdollisesti tuomia riskitekijöitä. Tästä huolimatta on mahdollista tehdä joitakin johtopäätöksiä. Ensinnäkin voidaan todeta, että itse ohjelman toteuttaminen ja sen implementoiminen verkkoon testausta varten ei ollut Fabricin dokumentaatiota seuraten erityisen haastavaa henkilökohtaisen kokemukseni perusteella. Älykkään sopimuksen käyttöönotto verkossa vaatii kuitenkin prosessin, joka ei ole tarpeen perinteisiä keskitettyjä järjestelmiä käytettäessä sovelluksen testaukseen, joten tutustumatta huolellisesti järjestelmän erityispiirteisiin ja prosessin kulkuun dokumentaation avulla, tämä ei olisi ollut mahdollista.

Toiseksi; esimerkkitoteutuksen taustalle kuvattiin tilanne, jossa finanssialan toimijalla oli haasteenaan keskitettyä palvelinarkkitehtuuria käyttävän nykyisen järjestelmän turvallisuus. Järjestelmä sisälsi henkilötietoja ja muita sensitiivisiksi luokiteltuja tietoja, joita ei voitaisi varastoida julkiseen lohkoketjuun. Tietoa haluttiin pystyä hakemaan ja suodattamaan anonyymisti tilastointia varten sekä yksittäin henkilön tunnisteella. Ratkaisuksi tarjottiin Hyperledger Fabricilla toteutettua järjestelmää, johon implementoitiin pyydetyt toiminnot toteuttava ketjukoodi. Vaikka toteutus olikin rajallinen, voidaan sen todeta toimineen eräänlaisena konseptitodistuksena vaadittujen ominaisuuksien ja

tarkoituksenmukaisen tietorakenteen soveltuvuudesta Fabric-verkossa käytettäväksi.

Kolmanneksi; voidaan myös todeta, että luottamalla Fabric-verkon perusarkkitehtuuriin, voimme samalla hyväksyä todeksi, että mikä tahansa järjestelmän kehittäjien ohjeistamien hyvien käytäntöjen mukaisesti verkkoon tallennettu ja varmennettu tieto olisi suurella todennäköisyydellä turvassa esimerkiksi verkkoon kuulumattomien tahojen vaikuttamisyrityksiltä. Kuitenkin on syytä huomioida, että testiverkossa käytettiin vain kahta vertaista samanaikaisesti yhdellä päätelaitteella, eli tieto ei ollut tosiallisesti lainkaan hajautettua. Tällaisessa tapauksessa siis esimerkiksi tuhoamalla kyseinen päätelaite voitaisiin tuhota samalla kaikki verkon sisältämä tieto, riippumatta sen suojauksen tasosta. Kuten luvussa 2.3. todettiin, keskeinen osa lohkoketjuteknologian turvallisuudesta muodostuu tiedon hajautuksen kautta. Kun pääkirjan sisältämä tieto on tallennettuna toisistaan aidosti riippumattomille päätelaitteille, mahdollisuus tiedon pysyvälle tai väliaikaiselle häiriölle saatavuudessa on hyvin epätodennäköinen. Käytännössä hyvä suoja voidaan saavuttaa jo määrällisesti varsin pienelläkin hajautuksen tasolla. Mikäli tässä olisi kyseessä ollut tuotantokäyttöön viety todellinen toteutus, verkkoon olisi myös liitetty useita vertaisia ja pyritty huolehtimaan, ettei niiden kaikkien toiminta ei ole riippuvaista toisistaan. Näin toimimalla voitaisiinkin todeta, että ainakin alkuperäinen ongelma kriittisen tiedon sijaitsemisesta keskitetyillä kolmannen osapuolen hallitsemisilla palvelimilla voitaisiin ratkaista lohkoketjuteknologian avulla.

## LÄHTEET


Aggarwal, S., Kumar, N. 2021. Chapter Eleven - Cryptographic consensus mechanisms Introduction to blockchain. Elsevier, Advances in Computers, Volume 121, 211-226. PDF-dokumentti. Saatavissa: https://doi.org/10.1016/bs.adcom.2020.08.011 [viitattu 22.4.2022].

Androulaki, E., Cachin, C., Ferris, C., Muralidharan, S., Murthy, C., Nguyen, B., Sethi, M., Stathakopoulou, C. 2018. Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. Association for Computing Machinery. PDF-dokumentti. Saatavissa: https://dl.acm.org/doi/pdf/10.1145/3190508.3190538 [viitattu 31.10.2021].

Apache. 2004. Apache license, version 2.0. The Apache Software Foundation. WWW-dokumentti. Saatavissa: https://www.apache.org/licenses/LICENSE-2.0 [viitattu 22.4.2022].

Back, A. 2002. Hashcash - A Denial of Service Counter-Measure. PDF-dokumentti. Saatavissa: www.hashcash.org/hashcash.pdf [viitattu 31.10.2021].

Barger, A., Meir, A., Manevich, Y., & Tock, Y. 2020. A Byzantine Fault-Tolerant Consensus Library for Hyperledger Fabric. IEEE International Conference on Blockchain and Cryptocurrency. PDF-dokumentti. Saatavissa: https://doi.org/10.48550/arXiv.2107.06922 [viitattu 31.10.2021].

Baset, S., Desrosiers, L., Gaur, N., Novotny, P., O'Dowd, A., Ramakrishna, V. Salman. A. B. 2018. Hands-On Blockchain with Hyperledger. Birmingham: Packt Publishing.

CGI. 2020. Mikä on lohkoketju? – Lohkoketjun anatomiaa. CGI Inc. WWW-dokumentti. Saatavissa: https://www.cgi.com/fi/fi/blogi/mika-on-lohkoketju [viitattu 26.3.2022].

Chaincoder. 2022. IDE for Hyperledger Fabric. Bernd Noetscher. WWW-dokumentti. Saatavissa: https://www.chaincoder.org [viitattu 22.4.2022].

Curl. 2022. Releases and Downloads. Curl Project. WWW-dokumentti. Saatavissa: https://curl.se/download.html [viitattu 22.4.2022].

Docker. 2022a. Get Docker. Docker, Inc. WWW-dokumentti. Saatavissa: https://docs.docker.com/get-docker/ [viitattu 22.4.2022].

Docker. 2022b. System requirements. Docker, Inc. WWW-dokumentti. Saatavissa: https://docs.docker.com/desktop/windows/install/#system-requirements [viitattu 22.4.2022].

Docker. 2022c. Virtualization. Docker, Inc. WWW-dokumentti. Saatavissa: https://docs.docker.com/desktop/windows/troubleshoot/#virtualization [viitattu 22.4.2022].

Dwork, C., Naor, M. 1993. Pricing via Processing or Combatting Junk Mail. Springer-Verlag Berlin Heldelberg. PDF-dokumentti. Saatavissa: https://link.springer.com/content/pdf/10.1007%2F3-540-48071-4_10.pdf [viitattu 31.10.2021].

Git. 2022. Downloads. Software Freedom Conservancy. WWW-dokumentti. Saatavissa: https://git-scm.com/downloads [viitattu 22.4.2022].

Ethereum. 2022a. Introduction to smart contracts. Ethereum Foundation. WWW-dokumentti. Saatavissa: https://ethereum.org/en/smart-contracts [viitattu 31.10.2021].

Ethereum. 2022b. Smart Contract Languages. Ethereum Foundation. WWW-dokumentti. Saatavissa: https://ethereum.org/en/developers/docs/smart-contracts/languages [viitattu 31.10.2021].

Ethereum. 2022c. The history of Ethereum. Ethereum Foundation. WWW-dokumentti. Saatavissa: https://ethereum.org/en/history [viitattu 31.10.2021].

Euroopan parlamentin ja neuvoston asetus (EU) 2016/679.

Fontaine, C., Galand, F. 2007. A Survey of Homomorphic Encryption for Nonspecialists. EURASIP Journal on Information Security. PDF-dokumentti. Saatavissa: https://link.springer.com/content/pdf/10.1155/2007/13801.pdf [viitattu 10.11.2021].

Frisby, D. 2014. Bitcoin: The Future of Money? Lontoo: Unbound.

Haber, S., Stornetta, W. S. 1991. How to Time-Stamp a Digital Document. Springer-Verlag Berlin Heldelberg. PDF-dokumentti. Saatavissa: https://link.springer.com/content/pdf/10.1007/3-540-38424-3_32.pdf [viitattu 31.10.2021].

Hepp, T., Sharinghousen, M., Ehret, P., Schoenhals, A., Gipp, B. 2018. On-chain vs. off-chain storage for supply- and blockchain integration? Berliini: De Gruyter.

Honkanen, P. 2017. Lohkoketjuteknologian lupaus. Arcada Working Papers 1/2017. PDF-dokumentti. Saatavissa: https://urn.fi/URN:ISBN:978-952-5260-78-6 [viitattu 22.4.2022].

Hyperledger. 2019a. Case Study: Honeywell Aerospace creates online parts marketplace with Hyperledger Fabric. PDF-dokumentti. Saatavissa: https://www.hyperledger.org/wp-content/uploads/2019/12/Hyperledger_CaseStudy_Honeywell_Printable_12.12.19.pdf [viitattu 22.4.2022].

Hyperledger. 2019b. Case Study: How Walmart brought unprecedented transparency to the food supply chain with Hyperledger Fabric. PDF-dokumentti. Saatavissa: https://www.hyperledger.org/wp-content/uploads/2019/02/Hyperledger_CaseStudy_Walmart_Printable_V4.pdf [viitattu 22.4.2022].

Hyperledger. 2019c. Identity. Hyperledger Foundation. WWW-dokumentti. Saatavissa: https://hyperledger-fabric.readthedocs.io/en/latest/identity/identity.html#identity [viitattu 22.4.2022].

Hyperledger. 2020a. Case Study: The Right Rx for the Pharmaceutical Supply Chain? Ledger Domain's Hyperledger Fabric Solution. PDF-dokumentti. Saatavissa: https://www.hyperledger.org/wp-content/uploads/2020/09/Hyperledger_CaseStudy_LedgerDomain_Printable.pdf [viitattu 22.4.2022].

Hyperledger. 2020b. Hyperledger Fabric. WWW-dokumentti. Saatavissa: https://hyperledger-fabric.readthedocs.io/en/latest/whatis.html#hyperledger-fabric [viitattu 22.4.2022].

Hyperledger. 2020c. Permissioned vs Permissionless Blockchains. Hyperledger Foundation. WWW-dokumentti. Saatavissa: https://hyperledger-fabric.readthedocs.io/en/latest/whatis.html#permissioned-vs-permissionless-blockchains [viitattu 22.4.2022].

Hyperledger. 2020d. Pluggable Consensus. WWW-dokumentti. Saatavissa: https://hyperledger-fabric.readthedocs.io/en/latest/whatis.html#pluggable-consensus [viitattu 22.4.2022].

Hyperledger. 2020e. Smart Contracts and Chaincode. Hyperledger Foundation. WWW-dokumentti. Saatavissa: https://hyperledger-fabric.readthedocs.io/en/latest/smartcontract/smartcontract.html [viitattu 22.4.2022].

Hyperledger. 2021a. Case Study: Managing the Metal and Mining Industry's Supply Chain with Hyperledger Fabric. PDF-dokumentti. Saatavissa: https://www.hyperledger.org/wp-content/uploads/2021/12/Hyperledger_CaseStudy_KrypC_Minehub_Printable_121321.pdf [viitattu 22.4.2022].

Hyperledger. 2021b. Case Study: How Tech Mahindra Deployed Hyperledger Fabric for the Digital Transformation of Abu Dhabi's Land Registry. PDF-dokumentti. Saatavissa: https://www.hyperledger.org/wp-content/uploads/2021/02/HyperledgerCaseStudy_TechMahindra_022421.pdf [viitattu 22.4.2022].

Hyperledger. 2021c. Case Study: How we.trade Helps Businesses Grow With Digital Smart Contracts Powered by Hyperledger Fabric. PDF-dokumentti. Saatavissa: https://www.hyperledger.org/wp-content/uploads/2021/10/Hyperledger_CaseStudy_WeTrade_Printable.pdf [viitattu 22.4.2022].

Hyperledger. 2021d. Getting Started - Install. WWW-dokumentti. Saatavissa: https://hyperledger-fabric.readthedocs.io/en/latest/getting_started.html#getting-started-install [viitattu 22.4.2022].

Hyperledger. 2021e. Writing Your First Chaincode. WWW-dokumentti. Saatavissa: https://hyperledger-fabric.readthedocs.io/en/latest/chaincode4ade.html [viitattu 22.4.2022].

Hyperledger. 2022a. Deploying a smart contract to a channel. WWW-dokumentti. Saatavissa: https://hyperledger-fabric.readthedocs.io/en/latest/deploy_chaincode.html#deploying-a-smart-contract-to-a-channel [viitattu 22.4.2022].

Hyperledger. 2022b. Docs. WWW-dokumentti. Saatavissa: https://hyperledger-fabric.readthedocs.io/en/latest/ [viitattu 22.4.2022].

Hyperledger. 2022c. Fabric Gateway. Hyperledger Foundation. WWW-dokumentti. Saatavissa: https://hyperledger-fabric.readthedocs.io/en/latest/gateway.html#fabric-gateway [viitattu 22.4.2022].

Hyperledger. 2022d. hyperledger-fabric-ca Documentation. PDF-dokumentti. Saatavissa: https://readthedocs.org/projects/hyperledger-fabric-ca/downloads/pdf/latest/ [viitattu 22.4.2022].

Hyperledger. 2022e. Ledger. WWW-dokumentti. Saatavissa: https://hyperledger-fabric.readthedocs.io/en/latest/ledger/ledger.html [viitattu 22.4.2022].

Hyperledger. 2022f. Prerequisites. WWW-dokumentti. Saatavissa: https://hyperledger-fabric.readthedocs.io/en/latest/prereqs.html#prerequisites [viitattu 22.4.2022].

Hyperledger. 2022g. Running chaincode in development mode. WWW-dokumentti. Saatavissa: https://hyperledger-fabric.readthedocs.io/en/latest/peer-chaincode-devmode.html [viitattu 22.4.2022].

Hyperledger. 2022h. Securing Communication With Transport Layer Security (TLS). Hyperledger Foundation. WWW-dokumentti. Saatavissa: https://hyperledger-fabric.readthedocs.io/en/latest/enable_tls.html [viitattu 22.4.2022].

Hyperledger. 2022i. Setup Logspout. WWW-dokumentti. Saatavissa: https://hyperledger-fabric.readthedocs.io/en/latest/deploy_chaincode.html#setup-logspout-optional [viitattu 22.4.2022].

Hyperledger. 2022j. Step Four: Commit the chaincode definition to the channel. WWW-dokumentti. Saatavissa: https://hyperledger-fabric.readthedocs.io/en/latest/chaincode_lifecycle.html#step-four-commit-the-chaincode-definition-to-the-channel [viitattu 22.4.2022].

Hyperledger. 2022k. Using the Fabric test network. WWW-dokumentti. Saatavissa: https://hyperledger-fabric.readthedocs.io/en/latest/test_network.html#using-the-fabric-test-network [viitattu 22.4.2022].

Hyppönen, M. 2021. Internet. Helsinki: WSOY.

IBM. 2021. How SSL works. IBM Corporation. WWW-dokumentti. Saatavissa: https://www.ibm.com/docs/en/i/7.1?topic=concepts-how-ssl-works [viitattu 21.3.2022].

ISACA. 2017. A View of Blockchain Technology From the Information Security Radar. ISACA Journal vol. 5. PDF-dokumentti. Saatavissa: https://www.isaca.org/-/media/files/isacadp/project/isaca/articles/journal/2017/volume-4/a-view-of-blockchain-technology-from-the-information-security-radar_joa_eng_0817.pdf [viitattu 21.3.2022].

Jabbar, A., Dani, S. 2020. Investigating the link between transaction and computational costs in a blockchain environment. Taylor & Francis Group, International Journal of Production Research, Volume 58, Issue 11. PDF-dokumentti. Saatavissa: https://doi.org/10.1080/00207543.2020.1754487 [viitattu 10.11.2021].

Jaeseung, L., Byungheon L., Jaeyoung J., Hojun S., Hwangnam K. 2021. DQ: Two approaches to measure the degree of decentralization of blockchain. ICT Express, Volume 7, Issue 3. PDF-dokumentti. Saatavissa: https://doi.org/10.1016/j.icte.2021.08.008 [viitattu 22.4.2022]

Jäntti, T. 2018. Mikä on lohkoketju ja mitä uutta se voisi tuoda lääkejakeluun? Lääkealan turvallisuus- ja kehittämiskeskus Fimea. PDF-dokumentti. Saatavissa: https://www.julkari.fi/handle/10024/136848 [viitattu 22.4.2022]

Kaasalainen, V. 2018. Lohkoketjuteknologian haavoittuvuudet. Jyväskylän yliopisto. PDF-dokumentti. Saatavissa: https://jyx.jyu.fi/bitstream/handle/123456789/59608/URN%3ANBN%3Afi%3Ajyu-201809214198.pdf?sequence=1&isAllowed=y [viitattu 31.10.2021]

Kemmoe, V. Y., Stone, W., Kim, J., Kim, D., Son, J. 2020. Recent Advances in Smart Contracts: A Technical Overview and State of the Art. IEEE Access, vol. 8, pp. 117782-117801. PDF-dokumentti. Saatavissa: https://doi.org/10.1109/ACCESS.2020.3005020 [viitattu 31.10.2021].

Lauslahti, K., Mattila, J, Seppälä, T. 2016. Älykäs sopimus – Miten blockchain muuttaa sopimuskäytäntöjä? Elinkeinoelämän tutkimuslaitos. PDF-dokumentti. Päivitetty 12.9.2016. Saatavissa: https://www.etla.fi/wp-content/uploads/ETLA-Raportit-Reports-57.pdf [viitattu 28.10.2021].

LinkedIn. 2022. LinkedIn News. LinkedIn Corporation. WWW-dokumentti. Saatavissa https://www.linkedin.com/posts/linkedin-news_theworkshift-economy-labormarket-activity-6887062336839016450-67iT [viitattu 22.4.2022].

Mitani, T., Otsuka, A., 2020. Traceability in Permissioned Blockchain. IEEE International Conference on Blockchain (Blockchain), 2019, pp. 286-293. PDF-dokumentti. Saatavissa: https://doi.org/10.1109/Blockchain.2019.00045 [viitattu 28.10.2021].

Nakamoto, S. 2008. Bitcoin: A Peer-to-Peer Electronic Cash System. Bitcoin Project. PDF-dokumentti. Saatavissa: https://bitcoin.org/bitcoin.pdf

NIST. 2022a. Availability. National Institute of Standards and Technology. Information Technology Laboratory, Computer Security Resource Center. WWW-dokumentti. Saatavissa: https://csrc.nist.gov/glossary/term/availability [viitattu 21.3.2022].

NIST. 2022b. Confidentiality. National Institute of Standards and Technology. Information Technology Laboratory, Computer Security Resource Center. WWW-dokumentti. Saatavissa: https://csrc.nist.gov/glossary/term/data_confidentiality [viitattu 21.3.2022].

NIST. 2022c. Data Integrity. National Institute of Standards and Technology. Information Technology Laboratory, Computer Security Resource Center. WWW-dokumentti. Saatavissa: https://csrc.nist.gov/glossary/term/data_integrity [viitattu 21.3.2022].

Npm. 2022. Getting started. Npm, Inc. WWW-dokumentti. Saatavissa: https://docs.npmjs.com/getting-started [viitattu 22.4.2022].

Nyyssölä, T., Paczkowski, M. 2020. Blockchain Technology and its utilization in Finnish companies. Jyväskylän ammattikorkeakoulu. PDF-dokumentti. Saatavissa: https://urn.fi/URN:NBN:fi:amk-2020121528306 [viitattu 31.10.2021].

Oscarson, P. 2003. Information Security Fundamentals. In: Irvine, C., Armstrong, H. (eds) Security Education and Critical Infrastructures. WISE 2003. IFIP Advances in Information and Communication Technology, vol 125. Springer. PDF-dokumentti. Saatavissa: https://doi.org/10.1007/978-0-387-35694-5_9 [viitattu 21.3.2022].

Qin, W., Rujia, L., Qi, W., Shiping, C. 2021. Non-Fungible Token (NFT): Overview, Evaluation, Opportunities and Challenges. Southern University of Science and Technology, logy 2 Swinburne University of Technology, University of Birmingham, CSIRO Data61. PDF-dokumentti. Saatavissa: https://doi.org/10.48550/arXiv.2105.07447 [viitattu 28.11.2021].

Rantala, J. 2018. Lohkoketjuteknologian yhteiskunta, Osa I: Bitcoinista Ethereumiin. Niin & näin -lehti. PDF-dokumentti. Saatavissa: https://trepo.tuni.fi/bitstream/handle/10024/105156/lohkoketjuteknologian_yhteiskunta_osa_I_2018.pdf?sequence=1 [viitattu 28.11.2021].

Salimitari, M., Chatterjee, M., Fallah, Y. (2020). A Survey on Consensus Methods in Blockchain for Resource-constrained IoT Networks. TechRxiv. PDF-dokumentti. Saatavissa: https://doi.org/10.36227/techrxiv.12152142.v1 [viitattu 31.10.2021].

Savelyev, A. 2016. Contract Law 2.0: «Smart» Contracts As the Beginning of the End of Classic Contract Law. National Research University Higher School of Economics. PDF-dokumentti. Saatavissa: http://dx.doi.org/10.2139/ssrn.2885241 [viitattu 10.11.2021].

Strehle, E. 2020. Public Versus Private Blockchains. Blockchain Research Lab. PDF-dokumentti. Päivitetty 30.9.2020. Saatavissa: https://www.blockchainresearchlab.org/wp-content/uploads/2020/05/BRL-Working-Paper-No-14-Public-vs-Private-Blockchains.pdf [viitattu 10.11.2021].

Szabo, N. 1994. Smart Contracts. WWW-dokumentti. Saatavissa: https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html [viitattu 31.10.2021].

Szabo, N. 1996. Smart Contracts: Building Blocks for Digital Markets. WWW-dokumentti. Saatavissa: https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html [viitattu 31.10.2021].

Szabo, N. 1997. Formalizing and Securing Relationships on Public Networks. First Monday. WWW-dokumentti. Saatavissa: https://doi.org/10.5210/fm.v2i9.548 [viitattu 31.10.2021].

Szabo, N. 2001. Trusted Third Parties are Security Holes. Satoshi Nakamoto Institute. WWW-dokumentti. Päivitetty 2005. Saatavissa: https://nakamotoinstitute.org/trusted-third-parties [viitattu 31.10.2021].

Szabo, N. 2008. Bit gold. Blogi-kirjoitus. Päivitetty 27.12.2008. Saatavissa: https://unenumerated.blogspot.com/2005/12/bit-gold.html [viitattu 31.10.2021].

Tani, A. 2020. Zero-knowledge proofs in blockchain applications. University of Helsinki. PDF-dokumentti. Saatavissa: http://urn.fi/URN:NBN:fi:hulib-202012084740 [viitattu 10.11.2021].

Telia Cygate. 2022. Ohjelmistokehityksen palvelut. Telia Cygate Oy. WWW-dokumentti. Saatavissa: https://www.teliacygate.fi/fi/neuvonta-ja-konsultointipalvelut/ohjelmistokehityspalvelut [viitattu 26.3.2022].

Themistocleous, M., Christodoulou, K., Iosif, E., Louca, S., Tseas, D. 2020. Blockchain in Academia: Where do we stand and where do we go? Institute For the Future, University of Nicosia. PDF-dokumentti. Saatavissa: https://doi.org/10.24251/HICSS.2020.656 [viitattu 22.4.2021].

Valtiovarainministeriö. 2019. Tutkimus: Lohkoketjuteknologian hyödyntämisestä liian suuret odotukset. Valtioneuvoston selvitys- ja tutkimustoiminta, tiedote 207/2019. WWW-dokumentti. Saatavissa: https://vm.fi/-//10616/tutkimus-lohkoketjuteknologian-hyodyntamisesta-liian-suuret-odotukset [viitattu 22.4.2022].

Visual Studio Code. 2022. Download Visual Studio Code. Microsoft Corporation. WWW-dokumentti. Saatavissa: https://code.visualstudio.com/download [viitattu 22.4.2022].

Wright, C. S. 2018. Turing Complete Bitcoin Script White Paper. nChain. PDF-dokumentti. Saatavissa: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3160279 [viitattu 28.10.2021].

Xu, M., Chen, X., Kou, G. 2019. A systematic review of blockchain. Financ Innov 5, 27. PDF-dokumentti. Saatavissa: https://doi.org/10.1186/s40854-019-0147-z [viitattu 10.11.2021].

Zetzsche, D., Douglas, A., Ross, B. 2020. Decentralized Finance. Journal of Financial Regulation, Volume 6, Issue 2. PDF-dokumentti. Saatavissa: https://doi.org/10.1093/jfr/fjaa010 [viitattu 28.11.2021].