Juha Koski

Enterprise Blockchains - Mitigating the Learning Path

Metropolia University of Applied Sciences

Bachelor of Engineering

Information and Communications Technology

Bachelor's Thesis

1 June 2020

Author Title Number of Pages Date	Juha Koski Enterprise blockchains - mitigating the learning path 49 pages + 7 appendices 1 June 2020
Degree	Bachelor of Engineering
Degree Programme	Information and Communications Technology
Professional Major	IoT and Cloud Computing
Instructors	Marko Uusitalo, Senior Lecturer

The enterprise blockchain technology for data storing is developing fast and it is moving from the experimental phase to a more real business use phase. The technology is quite complex and requires wire-area competences from the development tools and the technology concept. International surveys indicate that one of the obstacles for the technology adoption is the lack of the skills and knowledge of enterprise blockchain technology.

The purpose of this thesis was to find ways to lower the learning efforts for a person not familiar with the enterprise blockchain concept to learn the concept and to be able to start the application development with the technology. The requirement was that all the tools used for building a Proof-of-Concept application in this thesis must be open-source and that the application can be built with the devices available for thee private citizens.

There is no settled blockchain theory and the platforms differ from each other. Two widely used platforms Ethereum and Hyperledger fabric were chosen to illustrate the features for public and enterprise blockchains.

The practical part of the project was done by building a Proof-of-Concept end-to-end application for football player transfer registration business case. The Linux Foundation Hyperledger fabric enterprise blockchain technology was selected as a platform for the application. The Sample application named FabCar provided by Linux Foundation was taken as a template for the Proof-of-Concept application and the FabCar application was then modified for the Proof-of-Concept application needs.

The results of this study indicate that using a sample application as a basis for building an application and modifying it speeds up and encourages the development significantly when it is possible to get an application working rather quickly. This helps to gradually add and change elements in the application. The process taught that an enterprise blockchain application could be built with open-source tools.

It was also observed that the course offerings for the practical hands-on enterprise blockchain courses are quite scarce. As a suggestion, open Universities could offer courses where end-to-end applications with some enterprise blockchain platforms are built.

Keywords	blockchain, public blockchain, enterprise blockchain, Ethereum, Hyperledger fabric, Proof-of-Concept, learning

Tekijä Otsikko Sivumäärä Päiväys	Juha Koski Yritysten lohkoketjut – oppimispolun loiventaminen 49 sivua + 7 liitettä 1.6.2020
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikka
Ammatillinen pääaine	Verkot ja pilvipalvelut
Ohjaajat	Marko Uusitalo, Lehtori

Yritysten lohkoketjuteknologia kehittyy nopeasti ja on siirtymässä yrityksissä kokeiluvaiheesta todelliseksi alustaksi yritysten tarpeisiin. Teknologia on melko monimutkaista niin, että sovelluksen kehittämisen aloittaminen edellyttää laaja-alaista osaamista kehitystyövälineistä ja teknologiakonseptista. Kansainväliset tutkimukset osoittavat, että yksi esteistä teknologian omaksumiselle on osaamisen ja tiedon puute yritysten lohkoketjuteknologiasta.

Opinnäytetyön tarkoitus on etsiä keinoja alentamaan kynnystä yritysten lohkoketjun ymmärtämiselle ja sovelluskehityksen aloittamiselle sillä. Vaatimuksena opinnäytetyössä oli, että kaikki kokeilusovelluksen rakentamiseen käytetyt kehitysvälineet ovat avoimen koodin välineitä ja että sovellus voidaan rakentaa kaikille saatavilla olevilla laitteilla.

Lohkoketjuteoria ei ole vakiintunut ja teknologia-alustat eroavat toisistaan merkittävästi ominaisuuksiltaan. Näistä syistä päätettiin lohkoketjuteorian esittämisessä käyttää hyväksi laajasti käytössä olevia julkisen lohkoketjun alustaa nimeltään Ethereum ja yritysten lohkoketjujen alustaa nimeltään Hyperledger fabric.

Käytännön osuus opinnäytetyöstä toteutettiin rakentamalla päästä päähän kokeilusovellus (Proof-of-Concept) jalkapalloilijoiden pelaajasiirtojen rekisteröintiin. Sovellusalustaksi valittiin Linux Foundationin yritysten lohkoketjualusta nimeltään Hyperledger fabric. Linux Foundationin esimerkkisovellus (Sample) nimeltään FabCar valittiin pohjaksi kokeilusovellukselle. FabCar -sovellusta muokattiin sen jälkeen kokeilusovelluksen tarpeiden mukaisesti.

Opinnäytetyön loppupäätelmänä opittiin, että käyttämällä esimerkkisovellusta pohjana uuden sovelluksen rakentamisessa pystytään sovelluskehitystä nopeuttamaan ja rohkaisemaan merkittävästi, kun saadaan aikaiseksi toimiva sovellus nopeasti. Toimivaan sovellukseen on sitten helppo lisätä osia ja muokata sovellusta. Prosessi opetti, että toimiva yritysten lohkoketjusovellus pystytään rakentamaan käyttämällä avoimen koodin välineitä.

Prosessin aikana havaittiin, että käytännönläheisiä yritysten lohkoketjukursseja ei juurikaan ole tarjolla. Ehdotukseksi nousee sen pohjalta, että esimerkiksi avoimet yliopistot voisivat tarjota käytännönläheisiä kursseja päästä päähän sovellusten rakentamiselle yritysten lohkoketjualustoilla osaamisen levittämiseksi.

Keywords	lohkoketju, julkinen lohkoketju, yritysten lohkoketju, Ethereum, Hyperledger fabric, oppiminen

Contents

List of Abbreviations

AML	Anti Money Laundering. A set of laws, regulations, and procedures intended to prevent criminals from disguising illegally obtained funds as legitimate income.
API	Application Programming Interface. Computing interface for software applications which defines interactions between multiple software intermediaries via a set of routines, protocols and tools.
CLI	Command Line Interface. Text-based interface used for entering commands to instruct a computer.
DApp	Decentralized Application. A computer application that runs on a distributed computing system.
DLT	Distributed Ledger Technology. A digital system for recording the transactions of assets in which the transactions and their details are recorded in multiple places at the same time. Unlike traditional databases, distributed ledgers have no central data store or administration functionality.
EOA	Externally Owned Account. Represents identities of external agents (human agents or automated agents) in Ethereum Blockchain. They use public key cryptography to sign transactions which are sent to Ethereum network.
EVM	Ethereum Virtual Machine. Isolated runtime environment for running smart contracts in Ethereum Blockchain.
KYC	Know Your Customer. The process of a business identifying and verifying the identity of its clients.
P2P	Peer-to-Peer. A network of computers which are connected to each other.
PoC	Proof-of-Concept. A small pilot project to test the design idea or assumption.
PoW	Proof-of-Work. One of the consensus algorithms in Blockchain network.
SDK	Software Development Kit. A set of tools used for developing applications provided by hardware and software providers.

1 Introduction

Enterprise blockchain technology is gaining popularity as a data storing technology. It is still very young technology and just moving from the experimental phase to a more real business use. The technology is quite complex which in part leads to a situation where the lack of skills and expertise become an obstacle for the adoption of technology.

The purpose of this thesis is to find ways how to lower the learning efforts for a person not familiar with the enterprise blockchain concept to learn the concept and to be able to start the application development with the technology. In order to explore the ways to lower the learning efforts, a Proof-of-Concept end-to-end application is built for football player transfer registering. A Sample application provided by the Linux Foundation Hyperledger Project is taken as a template and the necessary parts of the Sample application are modified for the Proof-of-Concept use. The idea is to use as much as possible from the Sample application so that it would be possible to build a functioning end-to-end application as fast as possible. Once the application works, it would be easy to gradually develop it.

The requirement for the thesis is that all the tools used to build the Proof-of-Concept application must be open-source tools and the materials are required to be freely available on the internet. The Proof-of-Concept application is done with a typical computer available for the private citizens and it is done at home environment.

The purpose of this thesis is not to verify the enterprise blockchain principles such as peer-to-peer decentralization and access management via channels. They are assumed to be working.

When compared to original blockchain concept in context of the Bitcoin virtual cryptocurrency presented by Nakamoto in 2008, the part of the enterprise blockchains are not strictly speaking blockchains, but rather part of the larger concept Distributed Ledger Technology (DLT) which contains the blockchain concept. In this thesis the term blockchain is used to cover also the enterprise blockchains – such as Hyperledger fabric - which conceptually are rather DLTs.

Section two describes the methods and data sources used in this thesis.

Section three creates a review on international studies in order to explore the status of the adoption of the enterprise blockchain technology. Also, one of the most widely used enterprise blockchain platform is selected to be used as the Proof-of-Concept platform.

Section four defines the main principles of the public blockchain concept. The information for the public blockchain defined in section four is then used as a basis in section five to define the enterprise blockchain concept.

In section six a working Proof-of-Concept end-to-end application is built. Finally, section seven provides a summary of the study, and the conclusions presented.

2 Material and Methods

The blockchain concept is very young although the technologies behind the concept are much older. The first white paper concerning the Bitcoin cryptocurrency which outlined the public blockchain concept was published year 2008 by Satoshi Nakamoto. It was not called blockchain then but the basic principles were there. Year 2015 Ethereum virtual cryptocurrency By Vitalik Buterin was published. Ethereum was built on the same concept as Bitcoin was but it brought the concept of the smart concept which made it as a general-purpose blockchain for various business areas, not just virtual currency.

The blockchain for enterprises started to emerge after the Ethereum was published. The technology was developing quite fast and the development has not set yet. That means that the focus regarding the studies, articles and literature must be mainly set to the couple of previous years.

Section three presents a review on the international studies to search for the trends for the importance of the blockchain technology and for the obstacles facing the adoption of the technology. The studies used were required to made by the prominent research communities which has long history of publishing studies and researches. The surveys on which the studies were based on, were required to be comprehensive.

The blockchain theory in sections four and five were based on the searches from domestic and international databases provided by Metropolia University, google scholar search, general google search and the developer community sites regarding Ethereum and Hyperledger fabric platforms as follows:

- Ebook Central
- O’Reilly Safari Online
- SpringerLink
- EBSCO eBook Collection
- Ellibs
- Metcaf Library Community
- https://scholar.google.com/

The Proof-of-Concept application in section six was based on the Samples template provided by the Linux Foudation Hyperledger fabric project and various internet articles covering the topic.

3 Adoption of the enterprise blockchain technology

Based on the international researches and surveys the blockchain technology is gaining importance in companies world-wide. The estimated investments are growing and the technology is becoming a critical priority in companies. At the same time the lack of competence and expertise are becoming an obstacle for the adoption of the technology.

This chapter provides an overview on the technology adoption prospects. Also, the most widely used blockchain platform for the enterprise blockchain technology is chosen. It will be used later in this thesis for presenting the blockchain concept.

Deloitte is a global organization originally founded year 1845 which provides consulting, accounting, risk advisory and tax services world-wide in 150 countries and territories. Deloitte conducted a survey “Deloitte’s 2019 Global Blockchain Survey - Blockchain gets down to business” year 2019 for getting insight of blockchain technology adoption. The survey covered 1386 senior executives working in large companies in thirteen countries.

One of the observations in the survey was that blockchain adoption has begun shifting from experimental Proof-of-Concept phase toward the building of real business applications. 53 percent of interviewees saw that the blockchain technology reached a critical priority in their company in year 2019 which was 10 percent rise from year 2018 (Figure 1). [Pawczuk 2019: 4.]

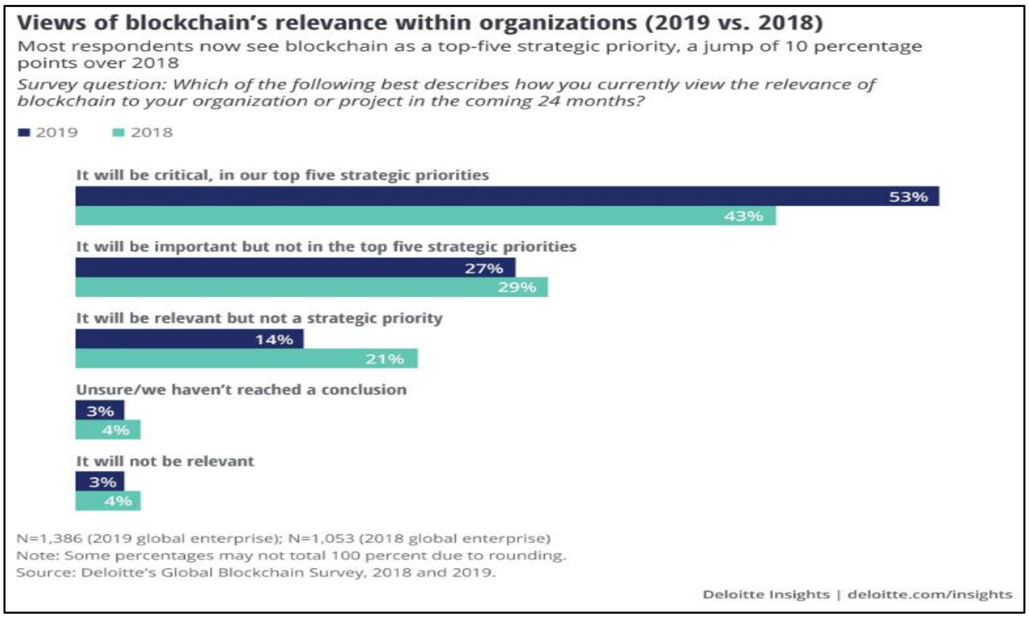


Figure 1. Deloitte Research survey for the relevance of the blockchain technology

At the same time the survey shows that 28% of interviewees saw that the lack of the inhouse skills and understanding is an obstacle for the technology adoption (Figure 2) [Pawczuk 2019: 39].

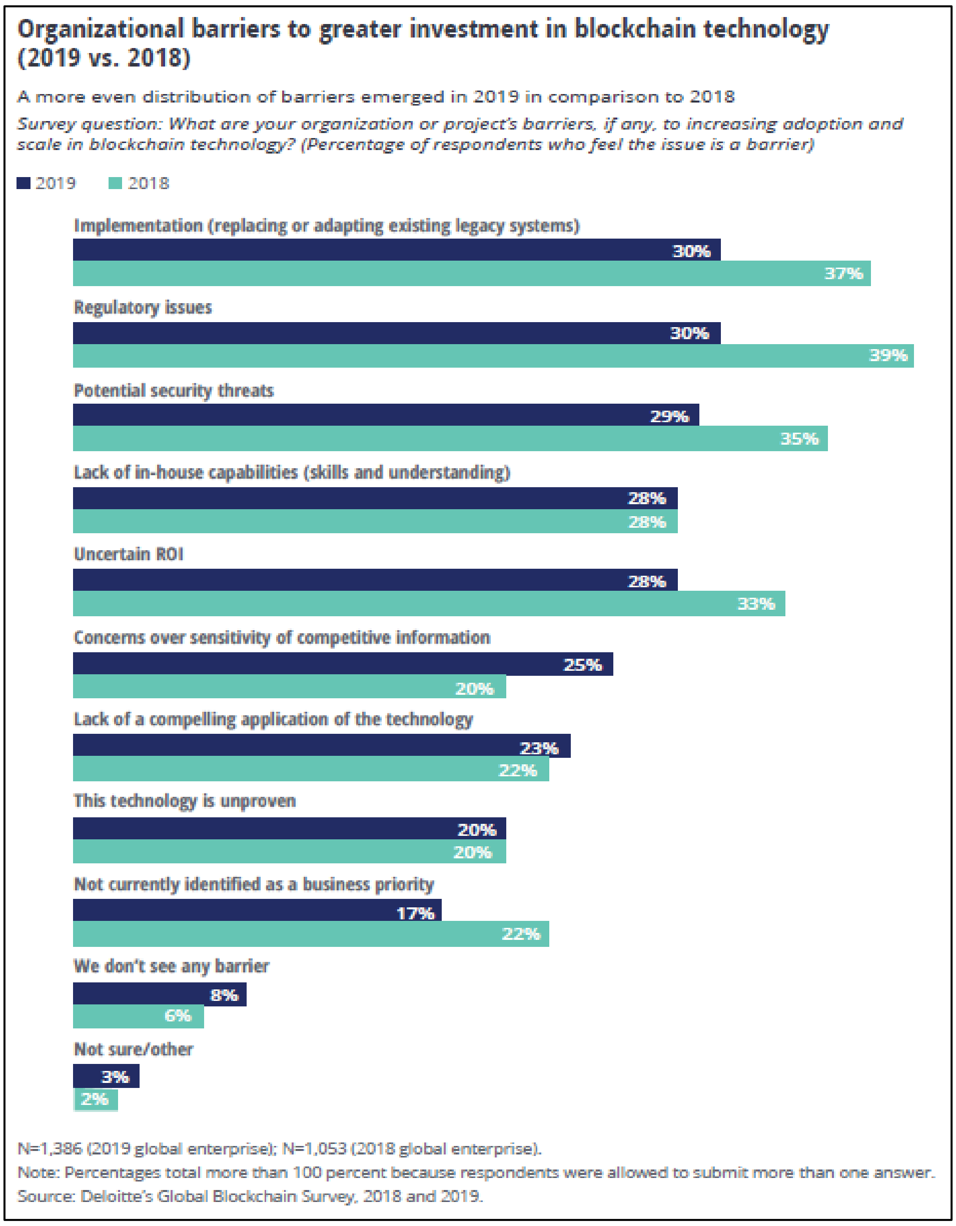


Figure 2. The barriers for the adoption of the blockchain technology by Deloitte survey

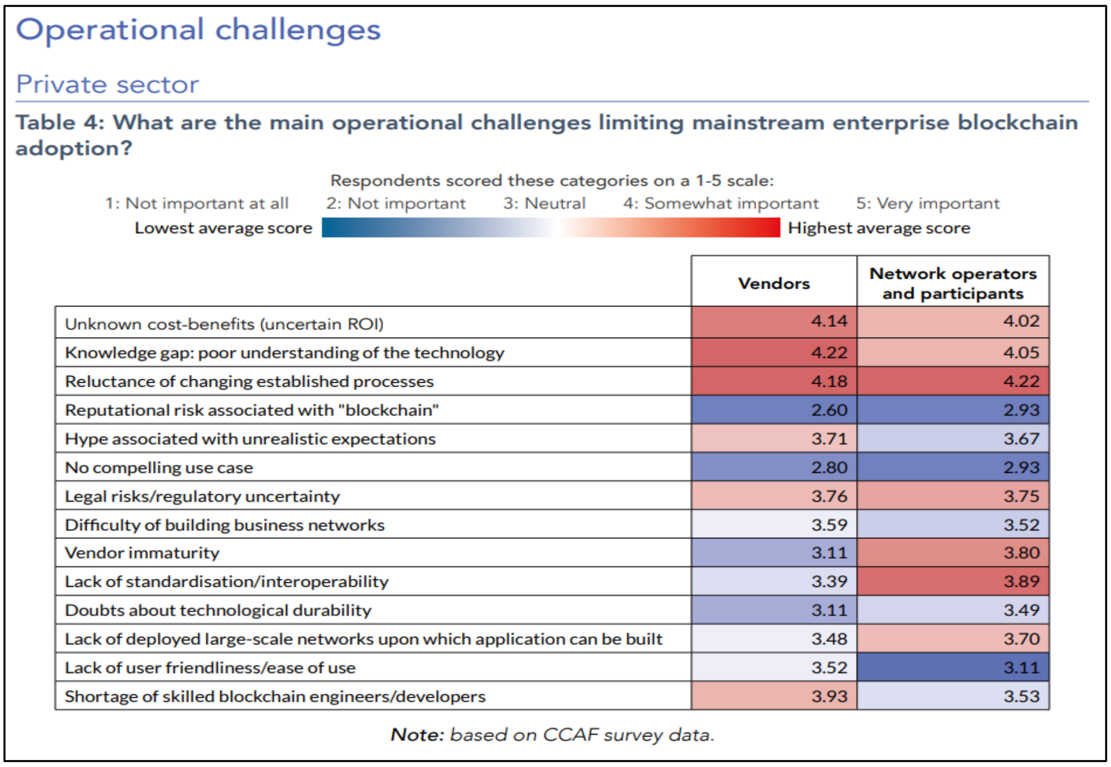
Year 2019 Accenture and World Economic Forum together in co-operation published a white paper “Building Value with Blockchain Technology: Is Blockchain Worth the Investment?” which was based on surveys of 550 persons across 13 industries and analysis of 79 blockchain projects.

In this study, the lack of expertise and understanding as one of the major risks for adoption of the blockchain technology was identified. The technology was seen to cause fundamental change in expertise which takes time and efforts to develop. [Warren & David 2019: 13.]

University of Cambridge published a study “2nd Global Enterprise Blockchain Benchmarking Study” of state of the blockchain technology adoption across the financial services industry. The study was based on survey data collected from more than 160 organisations from 49 countries and based on analysis of 67 live enterprise blockchain networks during July 2018 to June 2019.

The survey raised three main observations regarding the knowledge and expertise to be the main operational challenges in private sector which affect to the adoption of the blockchain technologies (Figure 3): [Rauchs etc. 2019: 70.]

- Knowledge gap: poor understanding of the technology
- Difficulty building business network
- Shortage of skilled blockchain engineers/developers



Operational challenges

Private sector

Table 4: What are the main operational challenges limiting mainstream enterprise blockchain adoption?

Respondents scored these categories on a 1-5 scale:
1: Not important at all 2: Not important 3: Neutral 4: Somewhat important 5: Very important
Lowest average score — Highest average score

	Vendors	Network operators and participants
Unknown cost-benefits (uncertain ROI)	4.14	4.02
Knowledge gap: poor understanding of the technology	4.22	4.05
Reluctance of changing established processes	4.18	4.22
Reputational risk associated with "blockchain"	2.60	2.93
Hype associated with unrealistic expectations	3.71	3.67
No compelling use case	2.80	2.93
Legal risks/regulatory uncertainty	3.76	3.75
Difficulty of building business networks	3.59	3.52
Vendor immaturity	3.11	3.80
Lack of standardisation/interoperability	3.39	3.89
Doubts about technological durability	3.11	3.49
Lack of deployed large-scale networks upon which application can be built	3.48	3.70
Lack of user friendliness/ease of use	3.52	3.11
Shortage of skilled blockchain engineers/developers	3.93	3.53

***Note:** based on CCAF survey data.*

Figure 3. The operational challenges for the blockchain technology adoption by Cambridge University survey

There is a wide variety of enterprise blockchain platforms available. This Cambridge University study data shows that Hyperledger fabric is currently the most popular (48%) of the covered live business blockchain networks be the platform of choice across all industries (Figure 4). The next biggest shares were taken by R3's Corda platform and Multichain platform.

The data from software vendors providing services on blockchain technologies available verify the number. The Hyperledger suite (where Hyperledger fabric is the main protocol) is mostly supported platform (53%) by development operators and the next is R3 Corda by its 35%. [Rauchs etc. 2019: 37.]

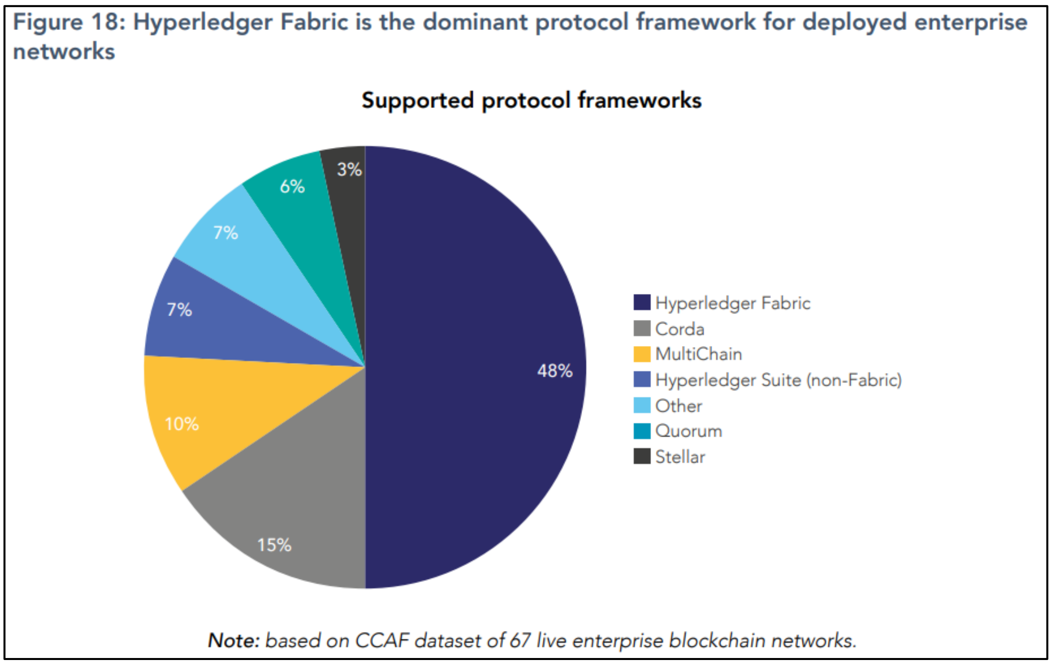


Figure 4. The survey for the market share of different enterprise blockchain platforms made by Cambridge University

The platforms for Hyperledger fabric and R3 Corda differ conceptually and technologically from each other very significantly.

4 Public blockchain

Public blockchain concept was first published in the white paper “Bitcoin: A Peer-to-Peer Electronic Cash System” describing the Bitcoin virtual cryptocurrency by Satoshi Nakamoto in year 2008. Bitcoin was originally fully meant for virtual transactions and it provided only a very limited scripting language. Application developers tried to adopt it for more general-purpose applications, but the effort was not successful due to the limited features of the Bitcoin scripting language. [Nakamoto 2008: 2-8.]

The basic Bitcoin principles are the following:

- It is open to everyone all around the globe without any permissions.
- The records stored in are time stamped and immutable via cryptographic methods so that it is not possible to change already accepted transactions afterwards.
- All transactions are transparent to all participants.
- All data is available for every participants.
- The consensus method ensures that the records added to the system are accepted by participants of the network.

Vitalik Buterin published in his white paper “A Next Generation Smart Contract & Decentralized Application Platform” in year 2013 a new platform called Ethereum. Ethereum used the Bitcoin principles as a basis and added the new smart contract feature in order to make it suitable for general application development. It is an open-source distributed platform based on the blockchain concept with added development functionality via smart contracts. Its virtual cryptocurrency is ETHER. [Buterin 2013: 1-34.]

Due to its general nature the Ethereum concept was selected for this thesis to be a tool to illustrate the blockchain principles for the public blockchain platforms. Ethereum is one of the most widely used public blockchain platform.

4.1 Lack of trust: Third-Party as a mediator required

In business transactions the parties involved do not fully trust each other. In order to solve the disagreements between the parties there need to be a settlement process including lawyers and audits to take care of the disagreements. Another option is to use a third-party verification authority which holds the necessary data to be able to verify and solve the disagreements. Both models are expensive and time-consuming, and the parties still have the trust problem. For example, buying a house requires trust between the buyer and the seller which can be ensured by employing the blockchain technology (Figure 5).

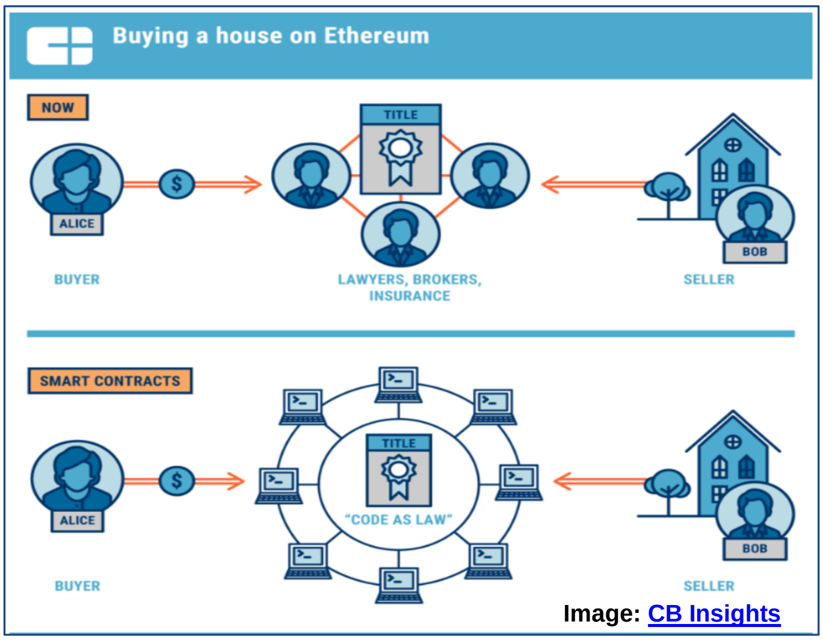


Figure 5. Blockchain as a trust mediator

The Blockchain concept enforces all data in business transactions in such a way that all parties can see it and ensures that no party is able to modify data after it is added to network thus providing trust between parties.

The blockchain trust promise is based on the following elements [Welfare 2019: 83-85]:

- **Transparency and openness** - every party can see the same data and are able to verify the software where blockchain is based on.
- **Immutability** - no party can modify the transaction data after its added to blockchain ledger.
- **Distributed** – every party has access to the same data and the same software.
- **Trusted** – removes the need for trusted third parties.
- **Decentralized governance** – no central authority controls the network.
- **Automated transaction handling** - provided via smart contracts.

In the centralized model (Figure 6) there is a central authority which controls the access to the system and owns the data stored there. It means that authority can close the system users of the system or the authority is able to sell users data causing challenges to privacy.

There is no way to be sure that the authority does not abuse or change the data in the system. This is problem for example in the developing countries where landowners cannot be sure that their ownership data is handled correctly and is not maliciously changed.

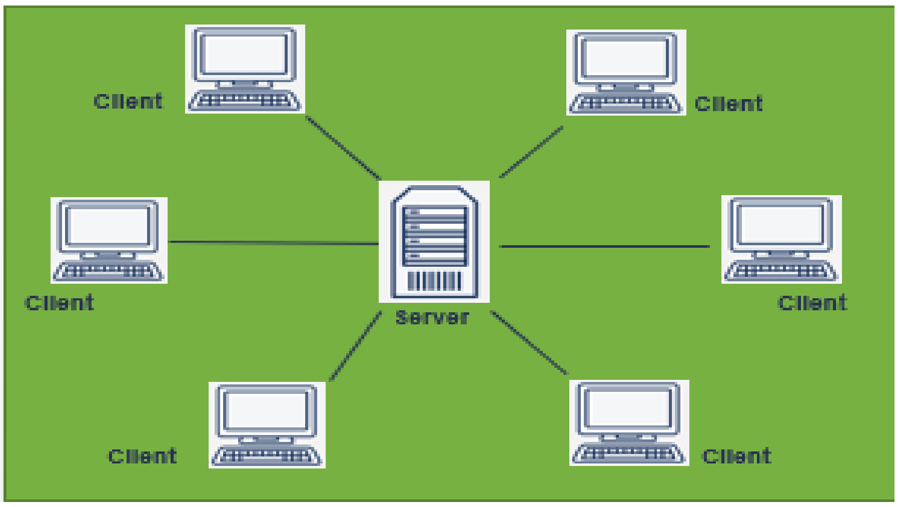


Figure 6. The centralized client-server model

In the decentralized peer-to-peer (P2P) model (Figure 7) there is no central authority which can control the system. Every participant has access to all data, can verify how the data is handled, can be sure data is not changed and no one can close participants off. Blockchain enforces this peer-to-peer model. For example, by using blockchain technology for the landowner records in developing countries, the landowners can trust that their ownership is safe.

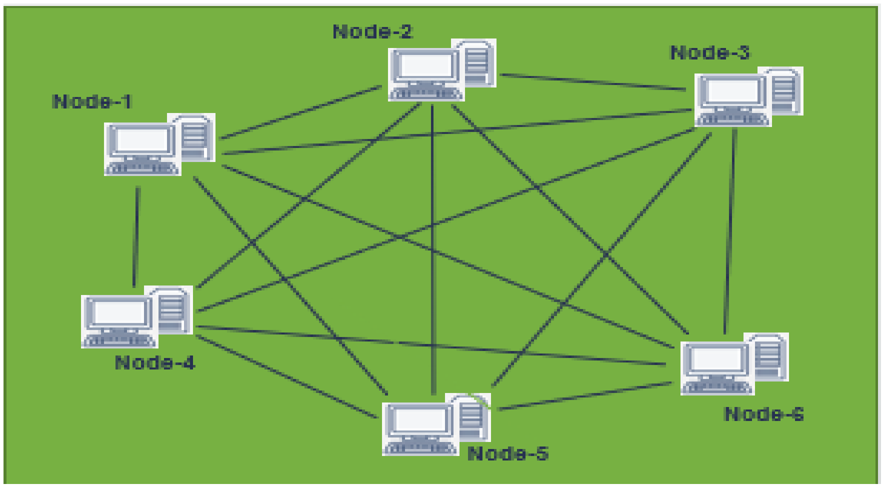


Figure 7. The decentralized peer-to-peer model

4.2 The basics of the blockchain technology

Blockchain is a distributed platform of trust which is append-on immutable, cryptographically secure, fully peer-to-peer shared to nodes, transparent and decentralized so that it is not controlled by any party. It is a technology to bypass the need for the central and third-party authority.

The transaction data is stored in chain-of-blocks structure (Figure 8) in chronology order. A predetermined number of transactions are collected to a block and are then accepted to the blockchain via mathematical consensus process called mining, where all willing participants can participate. Currently the consensus method in use in Ethereum is Proof-of-Work (PoW). It is based on complicated mathematical calculations where the mining incentive is Ethereum virtual currency ETHER. Due to the principles of the PoW consensus method, the throughput is only few transactions per second.

When a block is accepted to the blockchain via consensus process it is added to the end of the chain-of-blocks. The first block is Genesis Block. Every new block is connected to the previous block with a cryptographic pointer. The pointer is hash of the previous block. This cryptographic method ensures that the transaction data cannot be changed in the block which already has been added to the chain. Every modification to the transaction data causes that the hash for the block also changes and breaks the chain.

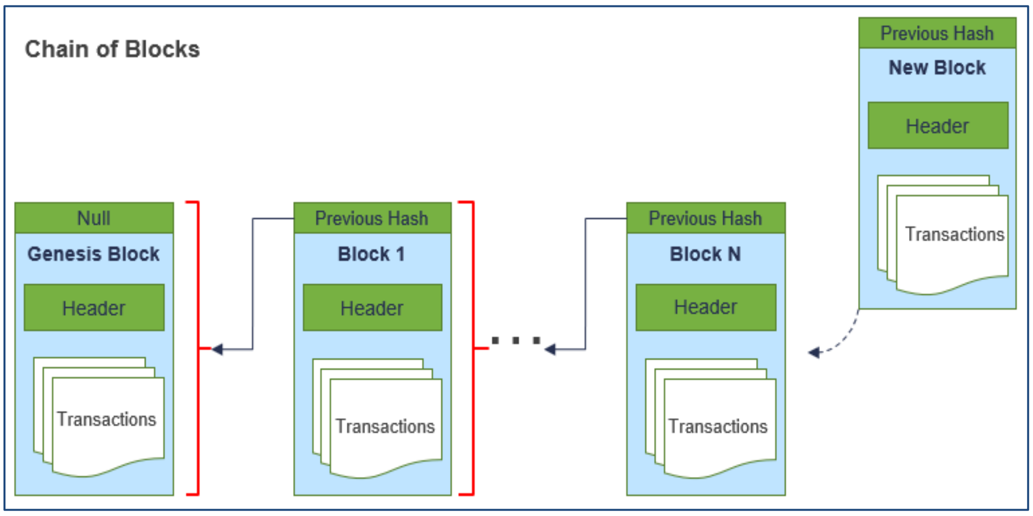


Figure 8. The immutable chain of blocks for storing transactions

The transaction data in the chain-of-blocks is replicated over the Peer-to-peer network to all participant nodes (Figure 9).

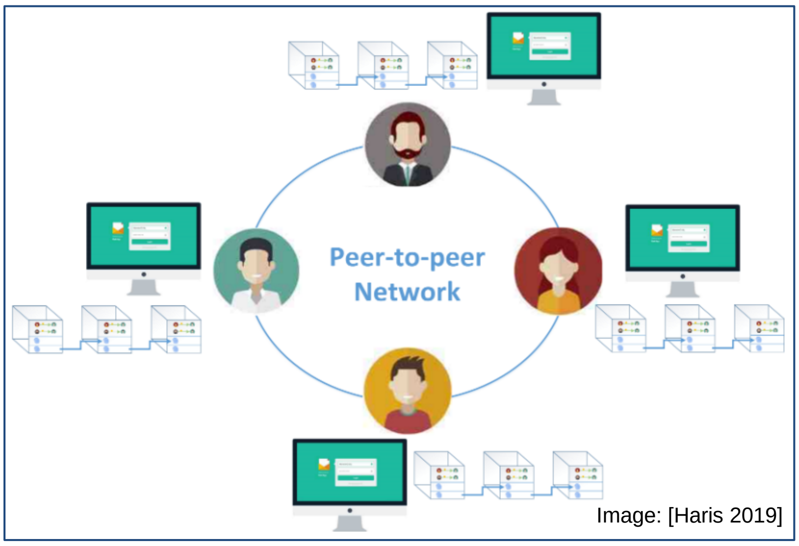


Figure 9. The replicated peer-to-peer model for the chain of blocks

The same applies also to the smart contracts and to the application data smart contracts handle and store (Figure 10). All nodes employ the same version of the blockchain software and smart contracts. All transactions are handled and every smart contract are run by Ethereum Virtual Machine (EVM) in every node. The application data in the blockchain is accessed only via smart contracts and the data is stored in a storage shared by each block in a node.

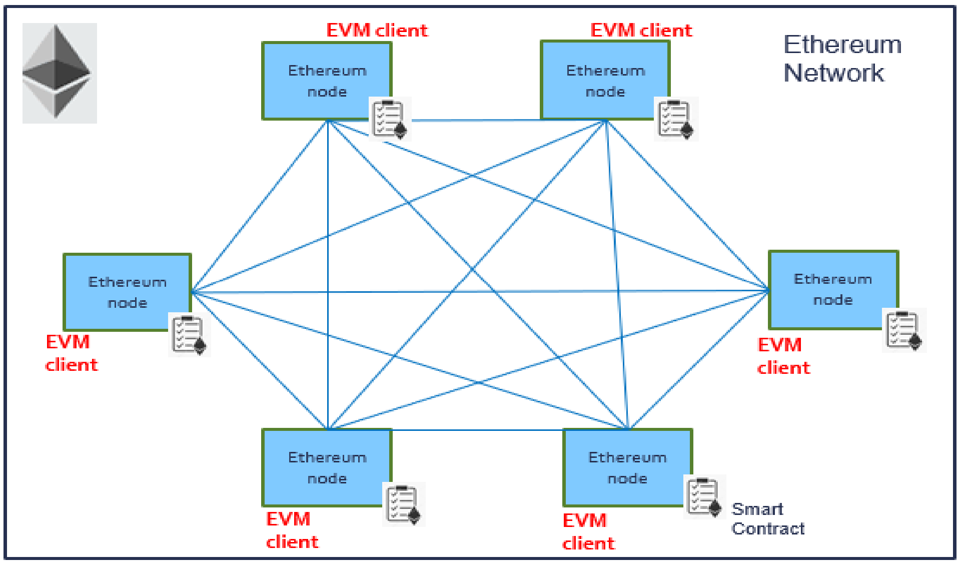


Figure 10. Smart contracts are deployed to every node in the blockchain network

4.3 Immutability in the public blockchain concept

Immutability in the public blockchain concept is enforced by two key technological methods – cryptographic hash functions and blockchain consensus mechanisms.

Cryptographic hash functions are used in the blockchain concept to build the chain-of-blocks structure by using the hash value of the previous block as a pointer to ensure that any changes made to the transactions in a block cannot be done undetected. Every change in the content of a block causes that hash value of the block changes and the chain in the chain-of-blocks structure breaks.

Consensus mechanisms – also called the mining methods in Bitcoin and Ethereum – are used to add new transactions collected in blocks to the chain-of-blocks structure and ensure that the content of a block cannot be changed or that any bloc in the chain-of-blocks structure cannot be replaced by a new block. It would mean that each block in the chain-of-blocks structure after the replaced one should be mined again. It would require such an enormous computing power that it is not possible to do.

Consensus mechanisms are a way to reach the joint agreement among the participants of a blockchain network that a block can be added to the network in the business environment where participants do not know and trust each other and there is no central authority to enforce the consensus. There are a variety of consensus mechanisms implemented in different public blockchains. Bitcoin and Ethereum currently utilise the consensus method called the Proof-of-Work consensus mechanism.

Hash functions

Hash functions are cryptographic algorithms which are based on complex irreversible mathematical functions. The basic function of hash functions (Figure 11) is that a message of arbitrary length is given as an input to the hash function which then produces a fixed length hash value as an output. The output is always the same length for a given method. Hash functions are deterministic in a sense that for a given input, they always produce the same output. SHA-3 256 hash function, which produces 252 bits length hash value, is typically used in blockchain technology implementations. The original message is not possible to derive back from the output hash value. [Singhal etc. 2018: section “How Blockchain works -Cryptographic Hash Functions”.]

The method used in in Ethereum is called Ethash. It is a variant of the SHA-3 hash function and it also always produces a 256 bits length hash value.

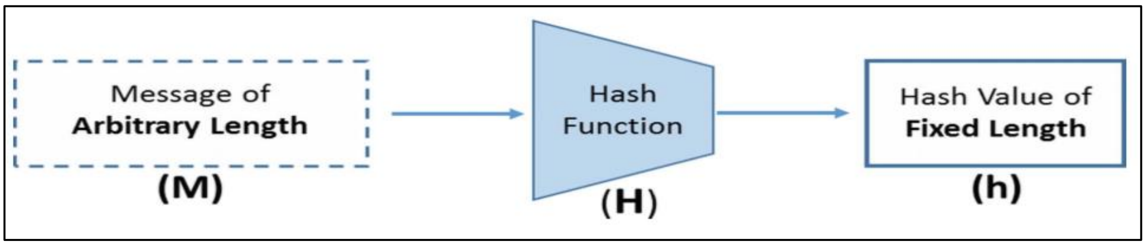
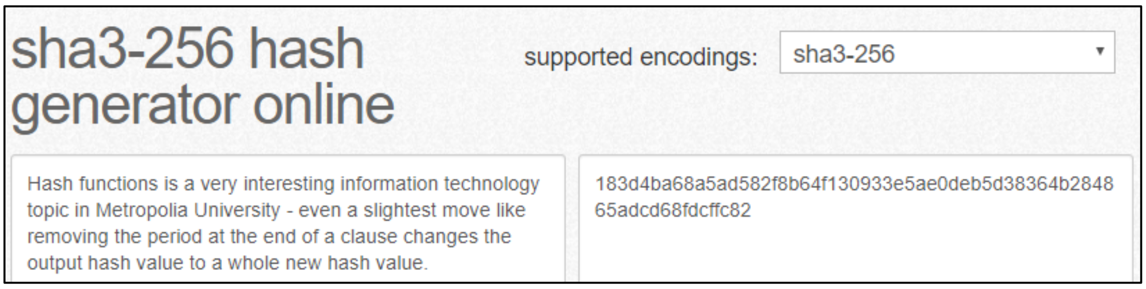


Image: [Singhal etc. 2018: section "How Blockchain Works - Cryptographic Hash Functions"]

Figure 11. Hash function basic functionality

Each modification to the input message causes that the output hash value will completely change. Figure 12 shows a message and the corresponding SHA3-256 hash value created by a SHA3-256 hash generator. The message is then modified by removing the point at the end of the message (Figure 13). As can be seen, the new output hash value is completely different.

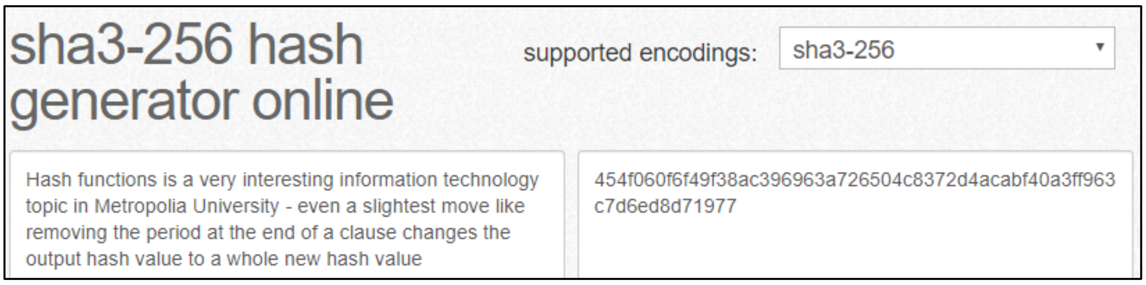


Hash functions is a very interesting information technology topic in Metropolia University - even a slightest move like removing the period at the end of a clause changes the output hash value to a whole new hash value.

183d4ba68a5ad582f8b64f130933e5ae0deb5d38364b2848
65adcd68fdcffc82

Image: [sha3-256 hash generator online]

Figure 12. The original message and the output hash value



Hash functions is a very interesting information technology topic in Metropolia University - even a slightest move like removing the period at the end of a clause changes the output hash value to a whole new hash value

454f060f6f49f38ac396963a726504c8372d4acabf40a3ff963
c7d6ed8d71977

Image: [sha3-256 hash generator online]

Figure 13. The modified message and the new output hash value

Proof-of-Work (PoW) consensus mechanism

The purpose of the Proof-of-Concept mechanism is to ensure that all nodes in the block-chain network agree on adding a new block to the network and that every node use the same blockchain. The mechanism also prevents the replacement of a block in the chain-of-blocks because it would require enormous amount of computing power.

The work of O'reilly, Dhameja Gautam and panda Priyansy Sekhar [Singhal etc. 2018: sections “How Blockchain Works” and “How Ethereum Works”] summarizes well the principles and use of the Proof-of-Concept” mechanism. The work has been used as a reference in the following description below.

Proof-of-Work consensus mechanism is the original blockchain consensus method firstly used in Bitcoin and Ethereum and then adopted to many other blockchain platforms. The method is also called mining and the nodes participating in mining are called miners. The method is quite ineffective, time-consuming and energy expensive. In addition to Proof-of-Work mechanism, there are variety of consensus mechanisms implemented in different blockchain platforms which are not based on the same principles as the Proof-of-Work mechanism to gain the joint agreement for adding a block.

The throughput in Ethereum is on average 15-30 transactions per second depending on the size of the blockchain network where a transaction compete with other transactions to get picked to a block, the complexity of the smart contract attached to the transaction and the configured difficulty level of the mining puzzle. In worst case scenario it may take minutes for a transaction to go through. In Ethereum the owner of the transaction pays a fee attached to the transaction record to get the transaction handled by a miner. The bigger fee the owner is ready to pay the faster the transaction goes through.

In Proof-of-Work (Figure 14) the miners are required to solve a difficult mathematical puzzle based on hash functions. The miners are competing against each other in the race to solve the puzzle. The first miner to solve the puzzle is rewarded with cryptocurrency implemented in the platform. In Ethereum platform the currency is ETHER. The method is asynchronous in a sense that it requires hard work to find the solution but the solution is easy to verify.

Each node in the blockchain network is free to participate in the mining process but due to the vast and expensive computer power required to reach the solution as a first miner, means that only the largest organizations can do the mining.

The mathematical puzzle is such that to solve the puzzle requires hard brute force. First the miners collect the transactions spread to every node into a block in a chronological order. The size of the block is predetermined thus limiting the number of the transactions collected to the block. The miners then run the header data filled in with a so-called nonce value through a hash function. The result of the run is a fixed length hash value. The hash value is then compared to a pre-set target value. The solution is found if the hash value is lower than the target value. If it is not, then the miner changes the nonce and tries again. This process continues until the hash value is lower than the target value or some other miner has found the solution first. The nonce value itself is a meaningless number. When the miner has found the solution, then the mined block is propagated to whole network and the solution is verified by every miner in the network. The verification is easy to do by rehashing the block header because the nonce in now known. When the verification process is ready, the miners drop their current mining task and start racing with a new block.

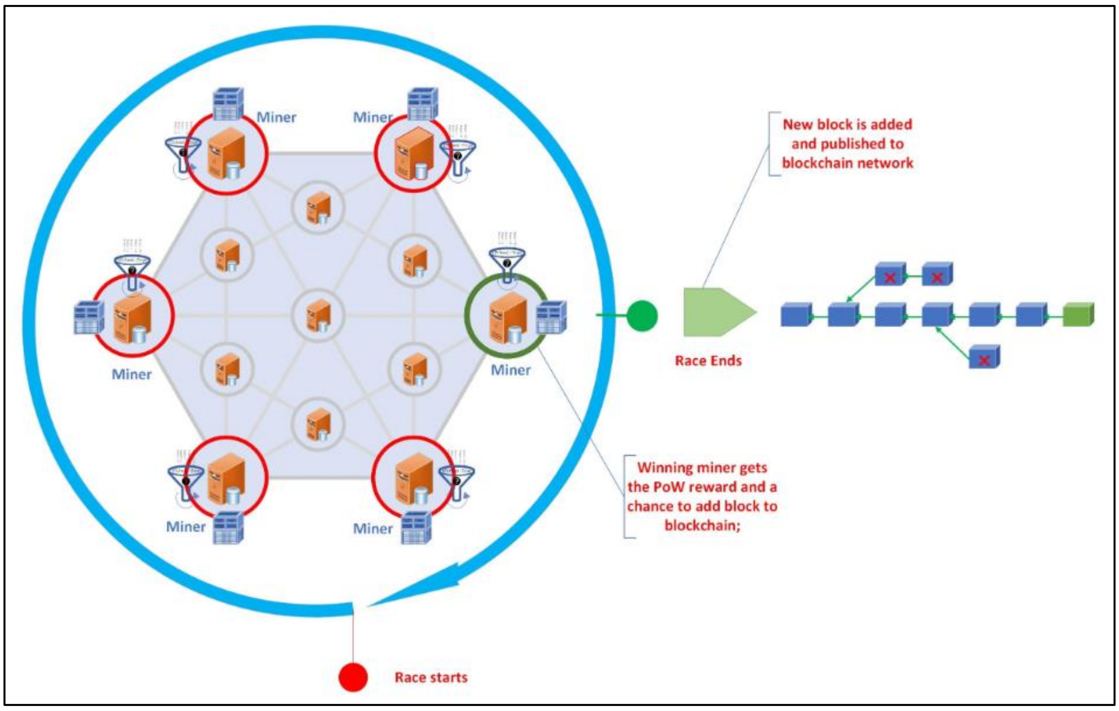

Miner
Miner
Miner
Miner
Miner
Miner
New block is added and published to blockchain network
Race Ends
Winning miner gets the PoW reward and a chance to add block to blockchain;
Race starts

Image: [Wu etc. 2019: section "Learn Ethereum – How PoW works"]

Figure 14. How Proof-of-Work works

The puzzle is formed in such a way that by tuning the target value, the solution time is kept in a predefined value. In Bitcoin the average time for finding the solution is set to be roughly ten minutes and in Ethereum it is set to be roughly 12 seconds. Real solution time may in Ethereum vary from seconds to even minutes.

4.4 Scalability

Public blockchains have inherent scaling problems. All transactions are propagated to all nodes and handled there. In Ethereum running the smart contracts causes additional latency. The larger the network is the more there are transactions congestion where transactions are competing with other transactions meaning more waiting time to get picked by a miner to get inserted into a block for mining. When the solution for a mining puzzle is found, the mined block needs to be propagated to each node in the network where it is verified.

In traditional computing the throughput challenges are typically solved by adding new servers to the server farm or the server cloud and tuning the load balancer to distribute the load evenly. However, in public blockchains the solution does not work. When new nodes join the network growing the size of the network, it means additional transaction congestion and transferring of transactions and mined blocks. The latency for a transaction going through grows when the size of the network grows.

The other scaling problem is that block mining time is kept in a predetermined value due to security reasons. The certain amount of computing is required to mine a block in order to ensure that the chain-of-blocks cannot be modified. When the block mining time differs from the predetermined target mining time, the value is modified so that mining time is always kept near the predetermined target mining time. In Ethereum the target mining time is roughly 12 seconds.

Visa is capable of processing tens of thousands payment transactions per second - Visa claims that its network is capable to handle 24.000 transactions per second - while Ethereum is capable to handle 15-30 transaction per second. There are efforts going on to improve Ethereum transaction throughput but so far, Ethereum is not suitable for high volume transaction handling. The time a transaction goes through in Ethereum may vary from seconds to minutes. That means that Ethereum is also not very suitable for example buying a cup of coffee from a coffee shop. Typical low volume business areas where Ethereum is used are for example supply chain management and land registrations.

The third scaling problem is that when the size of the network grows causing that also the size of the chain-of-blocks grows, it means the fewer miners have the resources to do the mining which in turn slows the mining process.

4.5 Ethereum Smart Contracts

The application data in the Ethereum blockchain is stored and handled only via smart contracts. A smart contract is a self-executing program which is written in Solidity programming language.

Solidity is an object-oriented programming language which is compiled to bytecode that is executable on the Ethereum Virtual Machine (EVM). The code controls the execution and executes when the specific conditions are met.

The code is injected to every node to be a part of a block in blockchain in the distributed, decentralized blockchain network (Figure 15). The code cannot be changed after it has been injected to the network.

The transactions are trackable and irreversible permitting the trusted transactions to be carried out among anonymous parties. Smart contracts are available for each participant after they have been deployed to the Ethereum blockchain network.

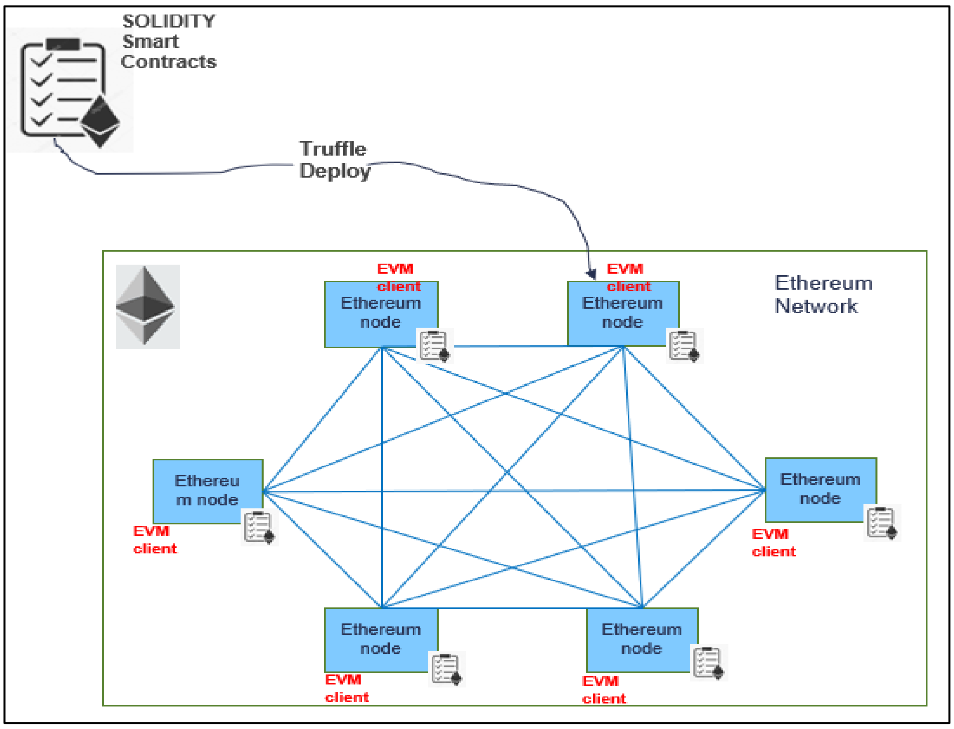


Figure 15. The smart contract code is injected into every node in the network

In Ethereum there are two kind of accounts – external user accounts (EOA) and contract accounts (Figure 16). EOAs are owned by users (or devices). The user IDs are based on the public key infrastructure and are anonymous. They can send transactions to other EOAs and activate smart contracts by sending transactions to contract accounts.

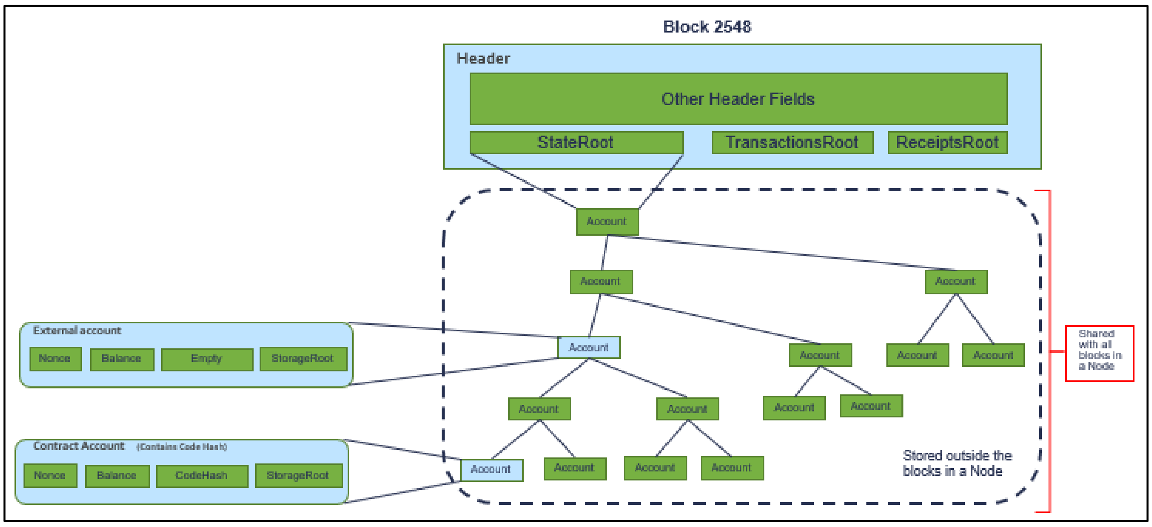


Image: Inspired by [Singhal 2018, 225]

Figure 16. The structure of a block in Ethereum node

The contract accounts contain code for the smart contracts and the application data. The application data is controlled only by the smart contracts. Both types of the accounts are stored in the same storage structure (state tree) which is stored in every node and shared by every block in a node.

The application data is stored inside contract account storage tree.

4.6 Ethereum Decentralized Applications (DApps) – development process

The applications made in Ethereum are called as decentralized applications (DApps). They consist of the client-side code (html, css and JavaScript) and the blockchain smart contract functionality.

Typical examples for DApps are thee applications such as e-Voting, Land Registration, Supply Chain, Stock Trading and Initial Coin Offering (Figure 17).

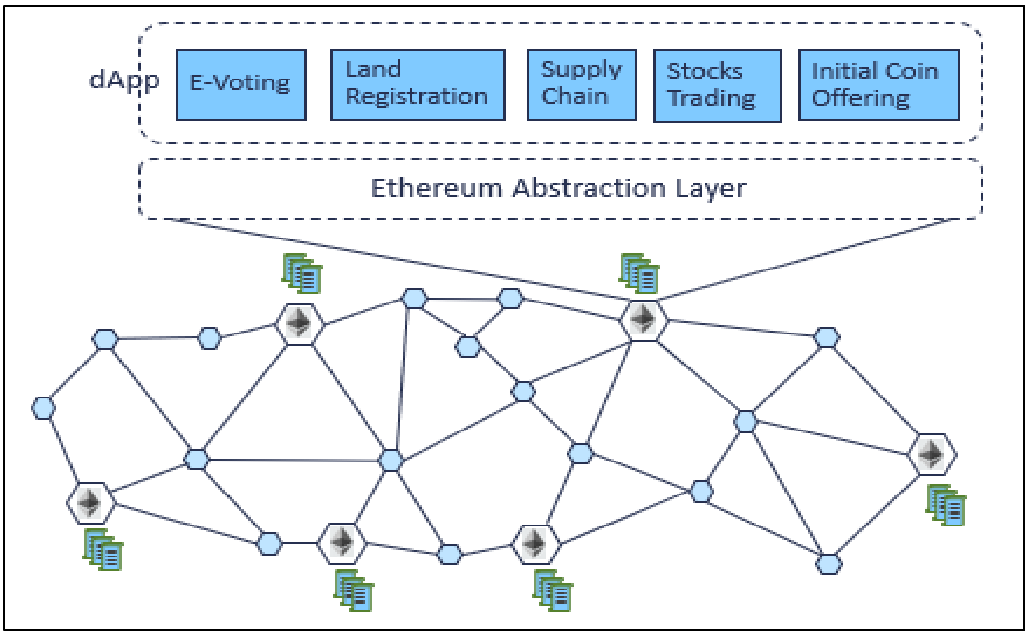


Image: [Singhal 2018, 221, slightly modified]

Figure 17. Typical decentralized applications in Ethereum blockchain

Various open-source developer communities have developed tools and tutorials which help in the Ethereum application development. One example for such a community is

Dapp University which provides different kind of full end-to-end applications as tutorials and templates [McCubbin 2020].

Figure 18 presents as an example some of the tools typically used in the Ethereum application development.

Truffle Suite [Truffle Suite] is an application development environment and testing framework for developing, testing and deploying smart contracts to Ethereum blockchain. Truffle Suite also provides Truffle Boxes templates. They are fully functioning end-to-end application templates which can be used a basis when building DApps.

One of the tools is Ganache [Ganache] which creates a simulated personal blockchain for testing purposes. It is very easy to start and using it for testing does require minimal technical knowledge of the underlying blockchain structure.

Metamask Wallet [Metamask] is a tool which helps the applications to connect to Ganache blockchains and via Infura APIs [Consensys] to a real Ethereum test blockchain networks.

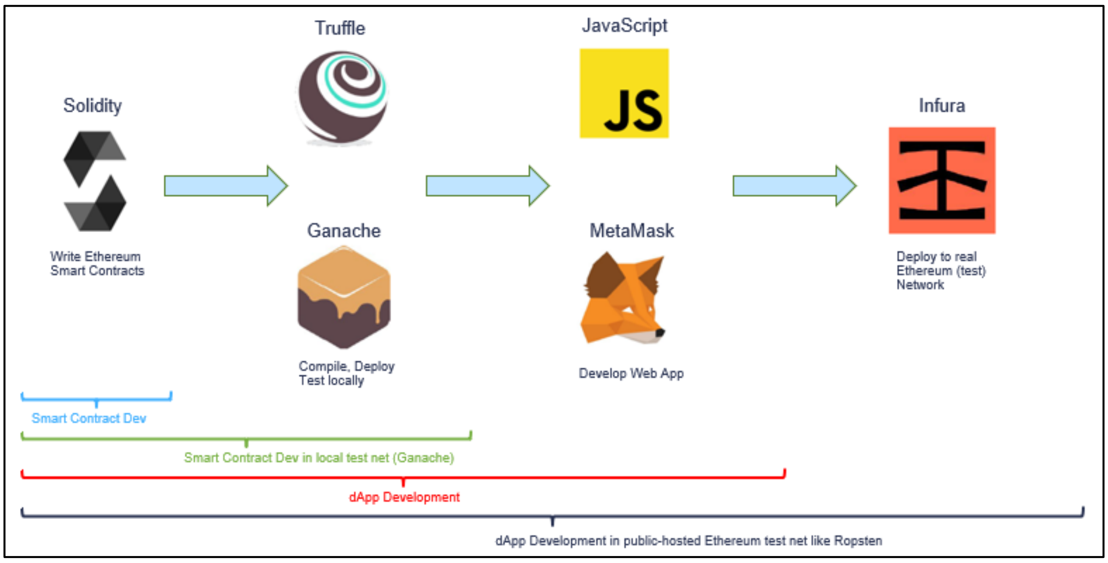

Solidity
Write Ethereum Smart Contracts
Truffle
Ganache
Compile, Deploy Test locally
JavaScript
MetaMask
Develop Web App
Infura
Deploy to real Ethereum (test) Network
Smart Contract Dev
Smart Contract Dev in local test net (Ganache)
dApp Development
dApp Development in public-hosted Ethereum test net like Ropsten

Figure 18. An example of the typical tools used in Ethereum application development

Figure 19 illustrates a typical Ethereum Dapp development process with development tools included.

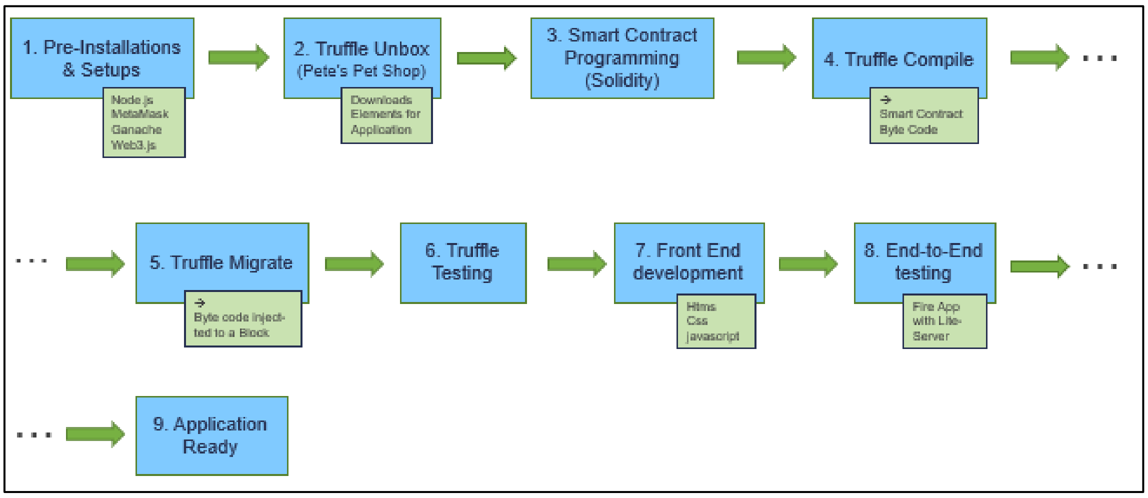

1. Pre-Installations & Setups
Node.js
MetaMask
Ganache
Web3.js
2. Truffle Unbox (Pete's Pet Shop)
Downloads Elements for Application
3. Smart Contract Programming (Solidity)
4. Truffle Compile
→ Smart Contract Byte Code
5. Truffle Migrate
→ Byte code injected to a Block
6. Truffle Testing
7. Front End development
Htms
Css
javascript
8. End-to-End testing
Fire App with Lite-Server
9. Application Ready

Figure 19. A typical application development process in Ethereum ecosystem

The Ethereum ecosystem consists of a quite large and active developer community which builds new tools and tutorials [Ethereum].

5 Enterprise blockchain

Public blockchains such as Bitcoin and Ethereum are open to anyone to join without needing to identify him/her to the blockchain network. After joining the user gets visibility to all data and the smart contracts in the network.

For enterprises to adopt the blockchain concept there are business goals and regulatory rules that they need to take in consideration and follow. Enterprises need to know with whom they are operating with. There are regulatory rules in financial transactions like Know Your Customer (KYC) and Anti-Money Laundering (AML). Many enterprise transactions require performance features that public blockchain technologies are not able to deliver yet. The transaction throughput per second needs to be in thousands rather than 15 seconds as in public blockchains.

Enterprises typically want to restrict the visibility, use and update of the data to the parties involved. For example, an enterprise has ten suppliers which delivers the same kind of product to the enterprise. The enterprise negotiates with suppliers for the terms of the deal such as the price of the product which may vary between suppliers. The parties of a deal obviously prefer keeping the terms of the deal secret from other parties not involving in it.

Currently there is no established definition for the enterprise blockchains. The number of different enterprise blockchain platforms can be counted in hundreds and their concepts may differ significantly from each other. However, there are certain characteristics that are typical for many of the implementations. Many of the blockchain implementations are not strictly speaking blockchains in sense how the blockchain was defined in section 2 but rather they belong under the technical umbrella called Distributed Ledger Technology (DLT) where blockchain is a sub concept. For example, typically enterprise blockchains are permissioned and not fully decentralised compared to permissionless and decentralized public blockchains. However, in this thesis all of them are called with the term blockchains.

Due to lack of the established definition and the heterogenous field of blockchain platforms, Hyperledger fabric was selected as an instrument to illustrate the basics for the enterprise blockchain platforms. The criteria for the selection was that it is currently one of the most widely used enterprise blockchains as described in the section three.

In this section the basic features for the enterprise blockchain technology are outlined. The public blockchain technology concept presented in section four is taken as a basis for the illustration and the differences between the public blockchain technology and the enterprise blockchain technology are highlighted.

5.1 The categorization based on the blockchain network openness

In section four blockchain technology principles for public blockchains was presented. Originally the principles were published in Bitcoin white paper, although they were not named as blockchain then in the paper. The blockchain model was public, open to anyone and without permissions. Public blockchain model does not fit well as such for the enterprises for the privacy and efficiency reasons.

In this section two more categories – private and hybrid – for the blockchain platforms which are more workable for the enterprise use are discussed. This categorization is merely a directive for classifying different blockchain platforms. Different platforms locate on the axle between the fully private and fully public blockchains. [Welfare 2019: 37-43.]

Public Blockchains

Public Blockchains are public-facing, fully open and distributed platforms which can be accessed by anyone in the world. They require no permission to join and everyone can send transactions to the network and see them in the ledger if they were proved to be valid to be added to the network. Everyone can read and write to the network.

The transactions are sent to the network so that the identity of the sender remains anonymous. The identity of the sender cannot be revealed by the key the sender used when sending the transaction.

Due to open nature of the platform and the lack of trust caused by anonymity, the mechanisms to accept transactions to ledger and to ensure data security and integrity are much more complicated, slow and expensive than in other models. Due to these features the transaction throughput is typically significantly less than in other categories and the energy consumption much higher. Bitcoin and Ethereum are examples of public networks.

Private Permissioned Blockchains

Private Permissioned Blockchains are closed networks which a group of the co-operating parties have decided to use for communicating the business transactions. The parties are required to have permissions to join the network to access it and their identity must be known to the network.

The rules for the network and for the parties are agreed between parties owning the network. They have the central authority to control the rules and manage the network.

Private networks employ permissions on many levels. They control for example what data each party can read and write, who can manage the rules and the network, who can deploy a smart contract and who can invoke a smart contract.

Typically, the private networks have less parties than the public networks. The parties do not fully trust each other but because parties are required to identify themselves there is full audibility for the actions the parties make in the network. Due to the closed nature of the network and that the identity must be known, the mechanisms to accept transactions to the ledger and to ensure data security and integrity are lighter than in other models. Due to these features the transaction throughput is typically much higher than in other categories and the energy consumption much lesser.

An example of the private permissioned networks is Hyperledger fabric.

Hybrid Blockchains

Hybrid Blockchains combines the features of both the public and the private networks. While preserving the openness nature of a blockchain they may still limit a part of the network to be private.

For enterprise use the users are required to identify themselves for different business and regulatory reasons like Know Your Customer (KYC). Although the network is kept open for anyone, it would still be possible to require identity for every user. The network would be open but the users are not anonymous.

The implementation of a hybrid blockchain may vary so that the features of both the public and the private networks are combined in different ways.

5.2 Enterprise blockchain characteristics

Enterprise blockchains are usually either private permissioned blockchains or some hybrid combination of the features of the public and the private blockchain. The parties in public transactions do not trust each other, even though they would be the business units from the same company. The companies usually span over many countries and they may be very complex.

Antony Welfare [Welfare 2019: 68] lists four key characteristics for an enterprise blockchain organisation:

- **Global** – organisations span over many countries.
- **Scalable** – organisations transact with each other with large number of transactions which may need to scale up to millions of transactions.
- **Secure** – data they are dealing with is often private related to people and assets.
- **Speed and accuracy –** data need to be available at speed and it need to be accurate.

Anthony Welfare [Welfare 2019: 68-69] also lists the main points why to use an enterprise blockchain:

- Enable trust inside and between the parties in large global enterprises.
- Mitigate the need for the third parties to ensure the transactions between the participants. To use third parties to solve business disputes often is expensive and time-consuming.
- Minimize manual error-prone human processes inside and between parties.
- improve the auditability problems caused by the incompatibility of cross systems and the reconciliation challenges.
- Improve the real-time information visibility between and inside the parties.

- Mitigate the risks and costs for the fraudulent transactions.

The home pages of Linux Foundation Hyperledger fabric developer community [Hyperledger fabric. Introduction] summarizes concisely requirements that need to be considered when using a blockchain in an enterprise:

- Participants to co-operation must identify themselves.
- The access and actions in the network must be permissioned.
- Throughput must be high.
- The transaction confirmation latency must be low.
- Data and transaction privacy and confidentiality must be secured.

5.3 The background information for the Hyperledger fabric platform

The Hyperledger fabric platform is hosted by the Linux Foundation under the Hyperledger project [Hyperledger fabric 2019: Introduction]. The project started in 2015 and it covers many Hyperledger blockchain platforms and standards in addition to Hyperledger fabric. Their approach is to provide an open developer community to advance the development of the blockchain technologies for the enterprise use.

Hyperledger fabric employs the basic features of other blockchains such as Ethereum. It stores the transactions in immutable time stamped order to the ledger and the client applications access the ledger via the smart contracts. The objects in the smart contracts handle are stored in a separate database called world state.

The biggest difference compared to the public blockchains such as Ethereum is that Hyperledger fabric is private and permissioned. In order to join the blockchain network, the participants are required to identify themselves. Different kind of policies control the permissions what a participant can do. The access to data and smart contracts can be controlled.

5.4 The concept of Hyperledger fabric

Hyperledger fabric blockchain technology enforces the same core ledger concept as public blockchain Ethereum does. Ledger comprises of the blockchain B – “chain-of-blocks” - and the world state W (Figure 20). The world state stores the smart contract data conceptually the same way as Ethereum state tree does. Technically the storing methods differ significantly.

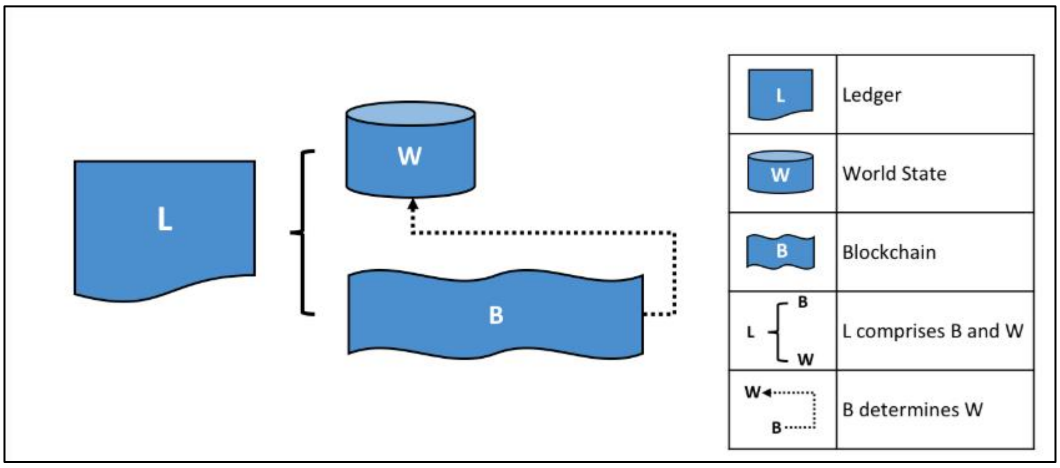


Image: [Hyperledger fabric 2019: Ledger]

Figure 20. Hyperledger fabric ledger structure

Hyperledger fabric is a private permissioned blockchain network where the co-operating organisations like enterprises or enterprise divisions agree to use Hyperledger fabric as a communication channel and agree on the rules of the co-operation. The rules are then implemented to the blockchain. The joining to network is based on invitation and the identity of each joining party is required to be known. A supply chain is a typical example for this kind of an arrangement.

Hyperledger fabric is permissioned blockchain. It is possible to configure the blockchain in such a way that only a subset of the parties communicates with each other for certain subjects. Other parties do not see the communication. There are also various mechanisms to define what each party and entity in the blockchain network can do.

Figure 21 presents a diagram [Hyperledger fabric. 2019: Basic network] for a small sample Hyperledger fabric **network** N which four **organisations** R1-R4 have decided and agreed on to exploit. In Figure 22 are the symbols of network named. R1-R3 are organisations which participate in business transactions. R4 is only for the administrational tasks. The sample blockchain network is governed by the policies agreed on between the organizations forming the network. The network policies are configured on the network policy NC4. The ordering service O4 works as an administration point for the network.

The core elements from the point of client application development are the following:

- **channels** C1 and C2
- **peer nodes** P1-P3
- **smart contracts** L1 and L2
- **client applications** A1-A3
- **certificate authorities** CA1-CA4
- **ledgers L1-L2**

Channel is the element which is formed by a subset of the blockchain network member organizations as a private way to communicate with each other so that the communication between the members is only visible to the organisations belonging to the channel. In the diagram there are two such channels C1 and C2. For example, channel C1 is formed by the organisations R1 and R2. Rules such as permissions which have been agreed upon on the channel are stored on the channel configuration CC1**.**

Peer nodes P1-P3 are connected to the channels so that one peer can be connected to several channels. For example, P1 and P2 are connected to channel C1. P2 is also connected to channel C2. The channel used defines the rules for the communication. Every channel has its own **ledger** which locates in every peer of the channel. For example, the ledger L1 locates both in peer P1 and P2.

Client applications in the organisations can access the ledgers only via **smart contracts**. Smart contracts are installed on some or all channel peers. They need not to be installed to all peers but only the peers which participate in the transaction acceptance process. The code of the smart contract in visible only on the peers where the smart contract is installed and hosted. For example, the smart contract S5 is installed on peers P1 and P2. The client applications A1 and A2 which belong respectively to channel C1 organizations R1 and R2 can call S5 to access the ledger L1 via peers P1 or P2.

Different elements use certificates to identify themselves to others as being from a certain organisation. **Certificate Authorities** CA1-CA4 provide the certificate services respectively for the organisations R1-R4. The certificates are also used to manage the permissions and to sign the transactions. See Figures 21 and 22 below.

When a client application calls a smart contract, it is required to identify itself via the certificate provided by Certificate Authority managing the certificates for the organization owning the client application. The client application is also required to pass the channel name, the peer name hosting the smart contract and the name of the smart contract.

In addition to acting as a network administration the **orderer** O4 is a core element in processing the transactions initialized by the client applications. The transaction handling process is not presented here because it is not essential to this thesis.

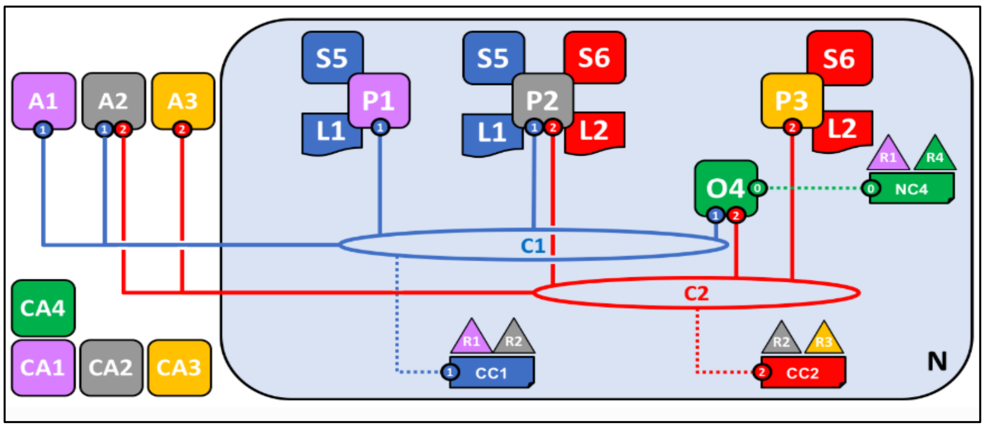
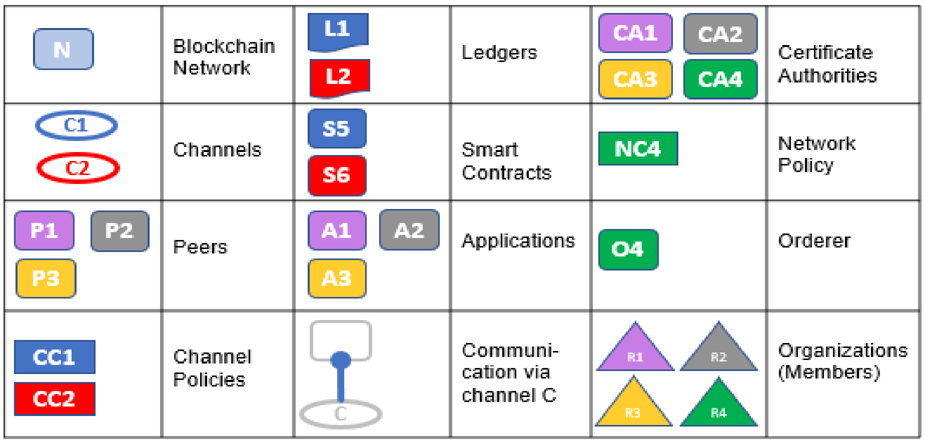


Image: [Hyperledger fabric 2019: Basic Network]

Figure 21. A small sample Hyperledger fabric network



N	Blockchain Network	L1 L2	Ledgers	CA1 CA2 CA3 CA4	Certificate Authorities
C1 C2	Channels	S5 S6	Smart Contracts	NC4	Network Policy
P1 P2 P3	Peers	A1 A2 A3	Applications	O4	Orderer
CC1 CC2	Channel Policies	C	Communi-cation via channel C	R1 R2 R3 R4	Organizations (Members)

Image: [Hyperledger fabric 2019: Basic Network]

Figure 22. The symbols in the sample Hyperledger fabric network

5.5 Immutability in the enterprise blockchain concept

In public blockchain concept there were two key features which ensured the immutability. The first was to use the hash values as pointers to build the chain-of-block structure and the second was the consensus mechanisms to reach an agreement among participants for a block to get added to the blockchain.

Enterprise blockchains implement the same hash method to build the chain-of-blocks structure by storing the hash value of the previous block to each block in the chain ensuring that the modifications to a block cannot be done undetected.

Regarding the consensus mechanism the enterprise blockchains differ from the public blockchains. In public blockchains the need for the consensus mechanism was derived from the fact that the participants in the network do not know and trust each other. The joint agreement for adding new block to the network and ensuring that the transactions are not modified needs to be done by technological means. The method used in Ethereum was Proof-of-Work.

In Enterprise blockchains typically participants know each other and also trust each other on some level. Additionally, participants are required to identify themselves to network leaving the audit trail thus ensuring that all actions in the network can be traced. In that environment the consensus can be reached by much lighter consensus mechanisms or it can even be omitted. The owners of the network make the rules what kind of consensus mechanisms are implemented and what actions are needed to reach the consensus.

Hyperledger fabric platform has been designed to allow the network owners to choose from variety of pluggable consensus mechanisms the one that is best suited for the business area and the relationships between the participants.

5.6 Scalability

The scalability in the enterprise blockchains depends heavily on the implementation of the platform. In this section scalability is presented via Hyperledger fabric platform.

A comprehensive laboratory study [Chung etc. 2019] was made where a group of researchers developed methodology and test environment to analyse the throughput of Hyperledger fabric platform. The throughput can be heavily affected in Hyperledger fabric implementation by the configuration of various elements and how the communication between elements is executed. Such elements are for example the database used for world state, Orderer configuration, endorsing policy, size of a block and channel policy to name few. The result for the transaction per second throughput was 1000 – 3000 transaction in a cloud computing environment [Chung etc. 2019: 44].

There are structural factors in Hyperledger fabric which affect and improve the scalability:

- Transactions are propagated only to the channel where the node handling them is attached, not to the whole network.
- Transactions are handled and smart contracts run only in the channel where the nodes handling them belong.
- The owner of the network decides the rules for the handling of the transactions.
- The consensus mechanism can be much lighter than in the public blockchains.
- The Ledgers are maintained per channel, not the whole network.
- Adding a new node does affect only the channel where it is attached, not the whole network.

The real use cases where enterprise blockchains have already been implemented are for example financial transactions, identity management, global shipping, inventory management and counterfeit product just to name few [Welfare 2019: 269-304].

5.7 Smart Contracts in Hyperledger fabric

Smart contracts are the client applications' view to the blockchain ledger, to query or update it, and are the only way to access the ledger. The Smart contracts initialize and manage ledger state through transactions which client applications submit. They are called "contracts" but they are just programs which represent application business logic.

In Hyperledger fabric, smart contracts related to each other are typically packaged as functions in the same chaincode file. The terms smart contract and chaincode are often used interchangeably but their purposes differ. The chaincode concept is merely needed for the smart contract deployment and it is mainly used by the administrators. The term smart contract represents the business logic and is merely used by application developers.

A smart contract accesses the both elements of the ledger – a blockchain which immutably stores the chain-of-blocks containing the transactions in the chronological order and a world state that keeps the current values of the smart contract objects which it handles.

The smart contracts are hosted in the peers. A smart contract can be installed in all or part of the peers belonging to the same channel. Only the peers where a smart contract is installed can see the code. After a smart contract has been installed to a peer, it is required to be instantiated to the channel where the peer is connected. That is how also the peers without installed smart contract get the interface to it. Only the peers participating in the transaction acceptance process regarding a smart contract, needs it installed. One peer can host many chaincodes.

Client applications invoke smart contracts to query or to update the ledger by sending a proposal transaction to a peer in the blockchain network.

Currently chaincode – or smart contracts from developers' point of view - can be written in one of the following languages: node.js, Java or Go.

5.8 Application development process in Hyperledger fabric

The application development with the Hyperledger fabric open-source tools is mainly based on scripting languages yet. There are no such tools available as in Ethereum ecosystem where there are for example the blockchain simulator Ganache and Truffle Suite for deploying and testing the smart contracts. An application development framework Hyperledger Composer has been developed by IBM Hyperledger project starting year 2016 to help modelling business networks above the Hyperledger fabric network and to help creating applications. However, when the Hyperledger fabric 1.4 was published on January 2019, the development of Composer was put in deprecated or on-hold status meaning that none of the developers in developer community are actively developing new features anymore. [Hyperledger Composer 2019.]

Hyperledger Composer

Hyperledger Composer is presented shortly below, although the development is not active anymore. The reason is that a large part of the literature handling the Hyperledger fabric hands-on application development process and which has been published before the release of the Hyperledger fabric version 1.4 uses the Hyperledger Composer as a tool to illustrate the process.

Hyperledger Composer is a model for application development offering a modeling language and APIs to define and deploy business networks above the Hyperledger fabric networks.

Hyperledger Composer tool uses four files [Figure 23]:

- Modelling file for business network elements
- Transaction functions
- Access control rules
- Query definitions

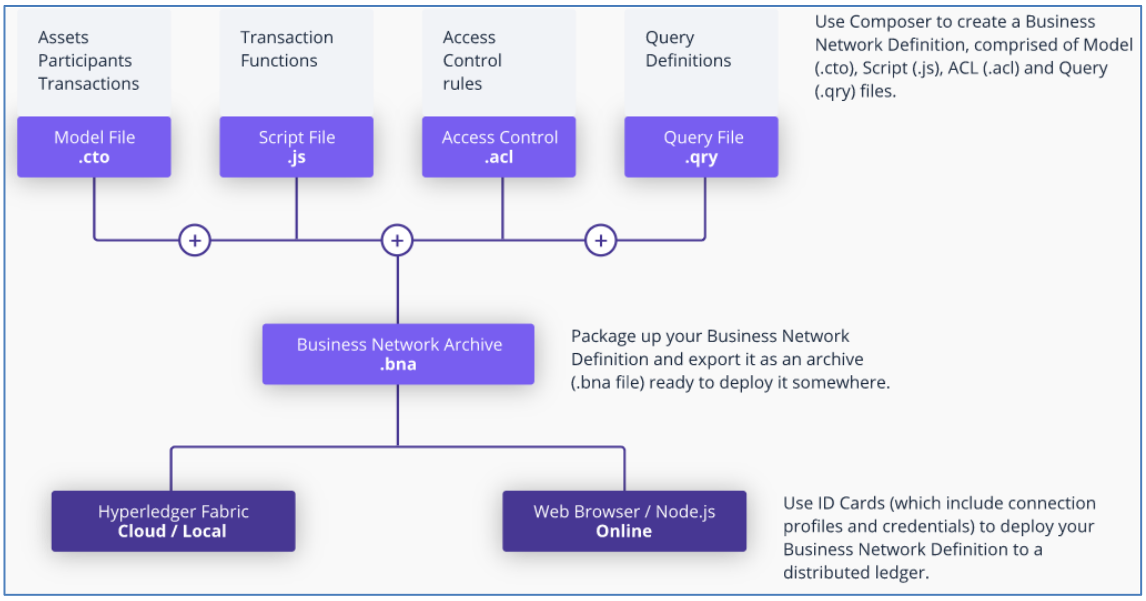

Assets
Participants
Transactions
Transaction
Functions
Access
Control
rules
Query
Definitions
Use Composer to create a Business Network Definition, comprised of Model (.cto), Script (.js), ACL (.acl) and Query (.qry) files.
Model File .cto
Script File .js
Access Control .acl
Query File .qry
Business Network Archive .bna
Package up your Business Network Definition and export it as an archive (.bna file) ready to deploy it somewhere.
Hyperledger Fabric Cloud / Local
Web Browser / Node.js Online
Use ID Cards (which include connection profiles and credentials) to deploy your Business Network Definition to a distributed ledger.

Figure 23. The concept of the Hyperledger Composer framework

The modelling file comprises of the following main business network elements:

- Assets (cars and listings)
- Participants (car buyers and owners)
- Transactions (buying and selling of cars)

The lead IBM Blockchain engineer Simon Stone IBM responsible for the Hyperledger Composer development gives three main reasons why the development of the framework has set to the on-hold status:

- The Composer is designed to support many of the blockchain platforms in the Hyperledger project. That has caused that there are two different programming models – one for the Hyperledger fabric chaincodes and one for the Composer business networks which causes confusion among the users and the need to update the both models.
- The dual design has made it a lot harder to adopt and expose the new features published for the Hyperledger fabric for the both programming models.
- It is challenging to maintain a sound API structure for building applications which really benefits the application development.

Client application basic flow

A client application interacts with a blockchain ledger by sending transactions to peers hosting the smart contract or querying the ledger. The application interacts with the ledger using Software Development Kit (SDK) which provides the services to connect the network and invoke the smart contracts. Currently SDKs for node.js and Java are available.

Figure 24 presents a simplified diagram for the steps how a client application invokes a smart contract to register a football player in an imaginary football player registration application. The client application submits the transaction which invokes the smart contract to register a player to blockchain.

Client application processes the transaction in six phases:

- Picks an identity from a wallet (class)
- Connects the blockchain network via a gateway (class).
- Accesses the desired blockchain network.
- Creates a transaction request to invoke a smart contract.
- Submits the transaction to the blockchain.
- Processes the response.

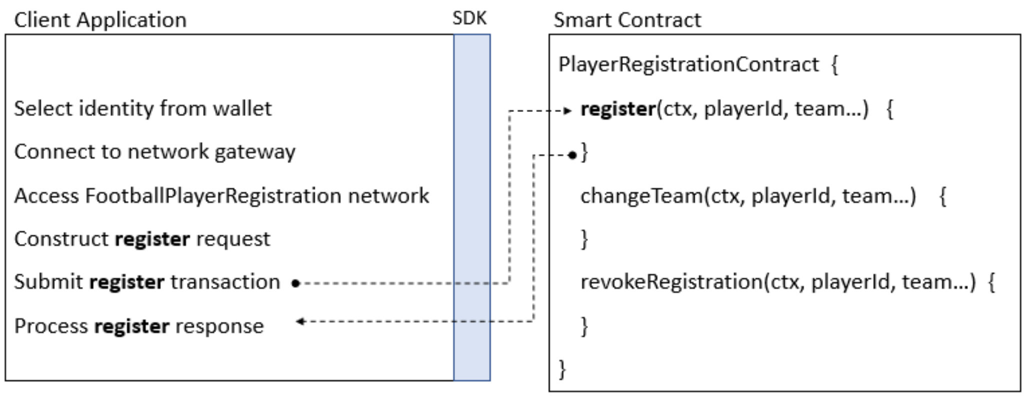

Client Application
SDK
Smart Contract
Select identity from wallet
Connect to network gateway
Access FootballPlayerRegistration network
Construct **register** request
Submit **register** transaction
Process **register** response
PlayerRegistrationContract {
register(ctx, playerId, team...) {
}
changeTeam(ctx, playerId, team...) {
}
revokeRegistration(ctx, playerId, team...) {
}
}

Figure 24. The steps for an application client invoking a smart contract

An example of an application development process with Hyperledger fabric

Figure 25 illustrates an example of an application development process (in local Hyperledger fabric blockchain) with open-source tools used in this thesis. An exception for the open-source principle was to use Windows 10 home in the development work but the principle was enforced by using virtual machine with Linux operating system as a guest. The blockchain network samples, binaries and docker images were provided by Linux Foundation Hyperledger fabric developer community [Hyperledger fabric 2019: Tutorials]. The tools in the parenthesis in the figure were available but were not used.

The open-source tools for building a Hyperledger fabric blockchain network and developing applications are in quite immature phase so that they are not very user-friendly and easy to use. The available tools are mainly based on scripts, tutorials and samples.

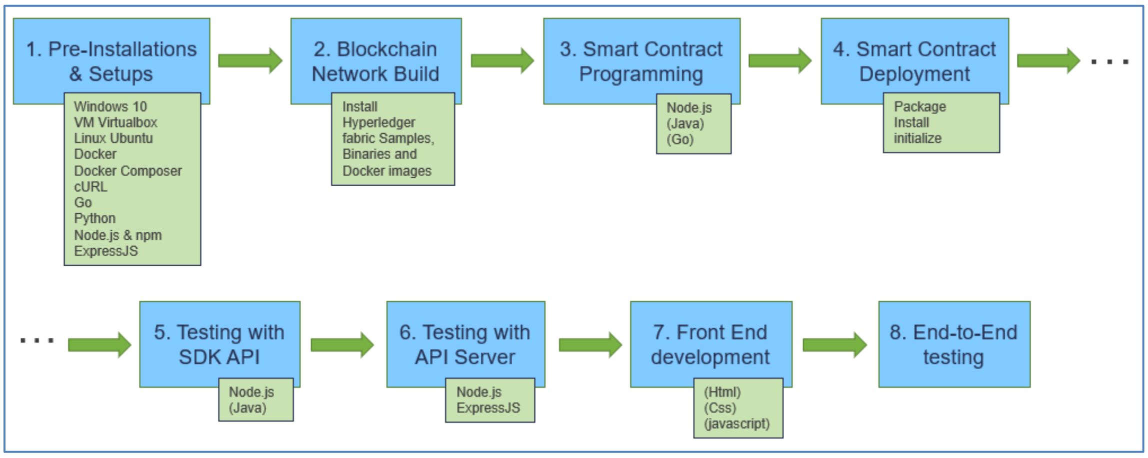


Figure 25. An example of an application development process in Hyperledger fabric ecosystem

Hyperledger fabric ‘dev mode’ for testing chaincodes.

Chaincodes are normally run and managed by the peers. Hyperledger fabric provides a mode called ‘dev mode’ where chaincodes are run by the user. This mode allows rapid code build, run and test cycle [Hyperledger fabric. 2019. Testing Using dev mode].

The “dev mode” leverages a pre-generated orderer and channel structures for a sample development network. This helps the smart contract developer to start faster the process of creating smart contracts and testing them. However, this feature was not employed in this thesis.

6 Proof-of-Concept (PoC): Football Player Transfer Registration

This section presents an end-to-end application as a simplified Proof-of-Concept. The purpose of Proof-of-Concept is to find as simple a way as possible to be able to understand the concept of enterprise blockchain and to start developing Hyperledger fabric end-to-end applications using the open-source tools.

The challenge with the enterprise blockchain application development is that as a young concept the blockchain configuration is still quite complex, user-friendly open-source development tools are not yet published and the supporting material like tutorials, guides, courses is quite scarce.

The preliminary condition in this thesis was that every device used are available for everyone regarding the price and the availability and all the software used are open-source.

The overall picture of the business case, its implementation, the development environment and main results from the test report are presented in this section. More detailed information can be found in the attachments.

All the code for Proof-of-Concept application is available in GitHub along the codes for selected parts of the FabCar Samples.

6.1 Business case – Football Player Transfer Registration

The business case for the Proof-of-Concept is the process for football players transfer registrations. There are many challenges in the international transfers. The participants do not trust each other for the records for a player to be correct. It may be difficult to collect the history of the player, especially if the player is not so famous yet. The delivery of the records usually takes a long time, months at best.

When a football player moves from one team to another international team there are chained actions required to be made in the chronological order to complete the transfer. The participants (in this PoC) are a player, his current team, the receiving team and the football association where the current team belongs.

The receiving team makes a transfer request for a player which is then verified by the player's current team. The current team checks that player has taken care of his/her obligations to the current team and is clear to move to the receiving team. Finally, the football association of the current team accepts the transfer when the current team has verified that all the obligations are in order.

The functions required are the following:

- Querying information for all players in the register.
- Querying information for one player in the register.
- Adding a player to the register.
- Make a transfer request for a player.
- Make a transfer verification for a player.
- Make a transfer acceptance for a player.

Figure 26 presents the transition state diagram how the transfer process proceeds from the state “Requested” to the state “Accepted”. Below the state rectangles are attached for each transition the records which are stored in the ledger in the chronological order.

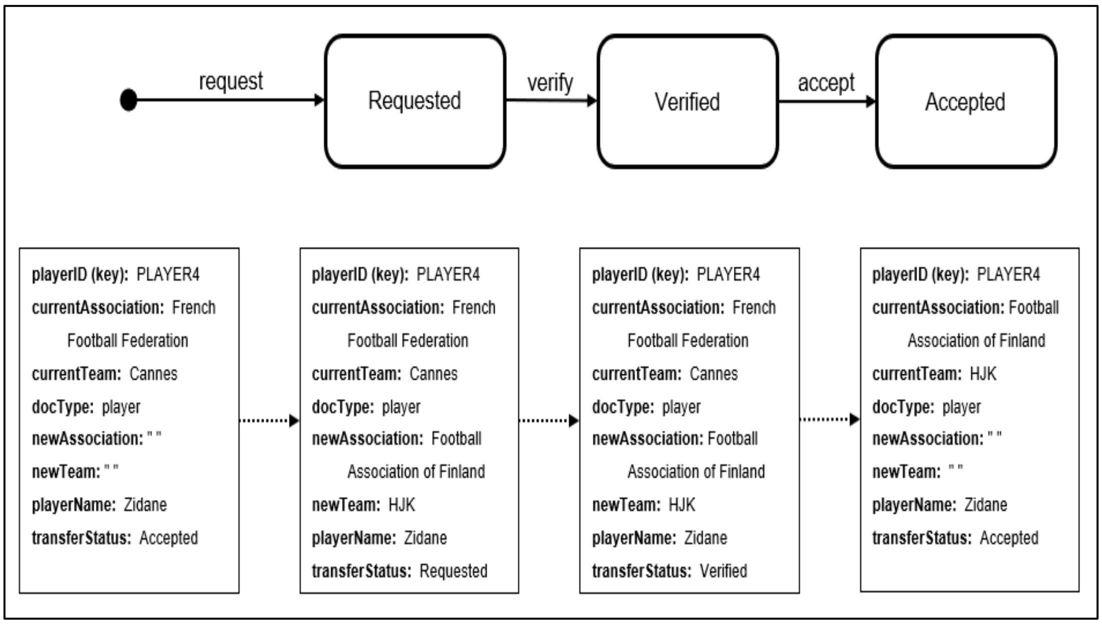


Figure 26. The transition state diagram for the football player transfer status changes

6.2 Proof-of-Concept implementation – Overall picture

Linux Foundation provides in its Hyperledger fabric pages [Hyperledger fabric 2019: Tutorials] samples ranging from simple one peer and channel network to a more complex end-to-end application. Samples covers binaries, codes and chaincode functions – smart contracts - to build the network and install/instantiate the chaincode functions.

One such end-to-end application is FabCar Car Registration application. The fully functioning network build with the ready-made scripts was named first-network. It also come with four smart contracts.

The Proof-of-Concept application was built by using the FabCar sample network and the smart contracts as a framework so that the new smart contracts were added to existing chaincode file fabcar.js and the new client application test scrips were created. The FabCar smart contracts were used to verify that the network itself works if there were problems with new PoC functions. The FabCar smart contracts were removed when the PoC was ready.

The PoC was implemented in the following phases:

1. The development environment was built.
2. The FabCar samples was installed and verified working.
3. The smart contracts for the PoC were created in the FabCar chaincode file.
4. The FabCar network was rebuild (with the PoC smart contracts included).
5. The PoC client application was tested with SDK node.js scripts.
6. The API Server (Express.js) was installed.
7. The PoC client application was tested with the API Server scripts.

6.3 The development environment

The requirement for this thesis was that all the tools used must be open source. The laptop used for the PoC was run on Windows 10 home which is not open source. For that reason, virtual machine Oracle VM Virtualbox was installed with Ubuntu 18.04 as a guest to simulate that the application was done wholly with open source tools.

Operating system

The operating system used in the PoC consist of four parts (Figure 27):

- Windows 10 home
- Oracle VM Virtualbox 6.1.2
- Ubuntu 18.04
- Docker engine version 18.09.7 and Docker-compose version 1.17.1

Five separate Docker containers were installed on Ubuntu guest system by the script which builds sample FabCar blockchain network (Appendix 1) in order to simulate a multi-peer Hyperledger fabric network.

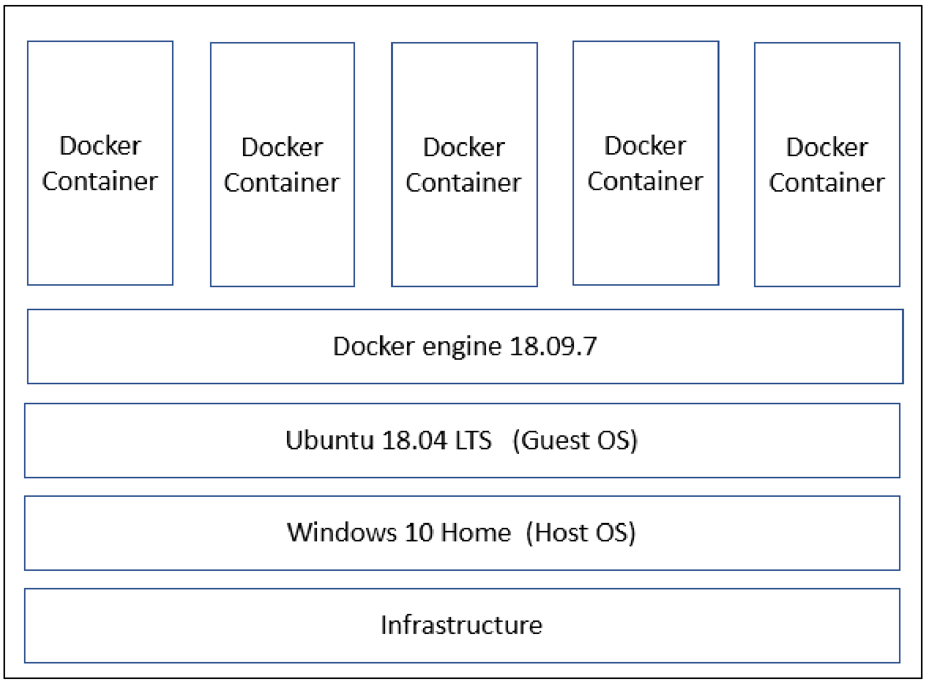


Figure 27. The operating systems elements used in the PoC

Installing prerequisites

The following tools were installed as a prerequisite:

- curl 7.58.0
- go version 1.13.7
- Node version 12.15.0
- Npm version 6.14.1
- Python version 2.7.17
- Visual Studio code with JavaScript support

6.4 The FabCar samples was installed and verified working

In the PoC was used the samples from the Linux Foundation Hyperledger fabric version 1.4.3. [Hyperledger fabric 2019: Introduction]. The newer version 2.0 was published on January 2020 [Hyperledger fabric 2020: Introduction]. It brought some improvements compared to earlier versions but no such new features that would have given added value to results and conclusions in this thesis so decision was made to stay in earlier version 1.4.3 due to stability reasons.

The FabCar application is a datastore which contains records for the registered cars. The records are stored in the world state. Every participant in the network has full access to the records and the data. All transactions which change the content of the FabCar world state fields are recorded to the FabCar ledger blockchain. Due to the cryptographic methods the stored transactions cannot be changed afterwards.

FabCar comes with the smart contracts to query the car information and update the data in the FabCar world state database. The smart contracts are the only way to access and change the data in the world state.

The data records contain the following fields:

- car Identifier (CarID, indexed by)
- maker
- model
- colour
- owner

There are five smart contracts in FabCar application:

- Network initialize with ten cars.
- Query for all cars.
- Query for one car.
- Car record creation.
- Owner change.

The first network consists of the following Hyperledger fabric elements (Figure 28):

- Two organisations, Org1 and Org2, which both come with the two peer nodes peer0 and peer1.
- One orderer organisation with one orderer using the SOLO ordering method.
- The world state databases for each peers are run in the separate CouchDB databases in each peer nodes.

- The organisations Org1 and Org2 come with the Certificate Authority which run fabric-CA with the required settings.
- The CLI command line interface for interacting directly with the fabric network without SDK. This can be used for directly testing the smart contracts without the SDK services. It was used for testing but he test were not included in this thesis because the tests made by SDK covered it.
- The Node.js client application implemented used node.js SDK services. The client application contains functions for creating the admin and application users, querying the car records and changing the owner.

All elements are run in containers in order to simulate separate devices for peers, orderer and Certificate Authority.

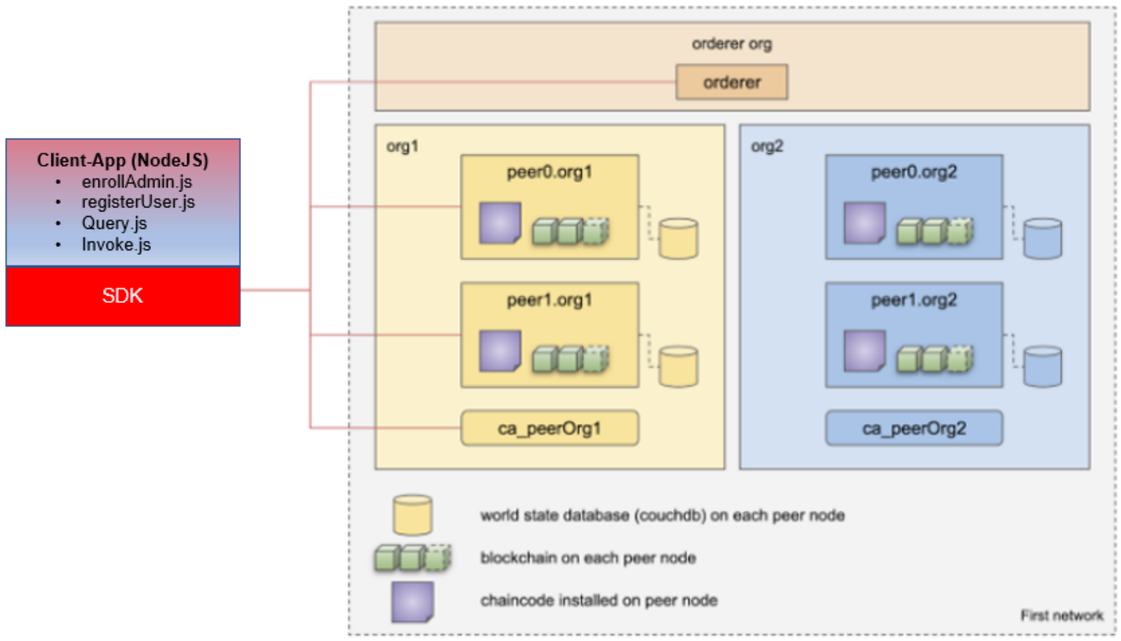


Image: [KC Tam 2019: 18]

Figure 28. The Hyperledger fabric network for the FabCar

The instructions for building the FabCar network and its smart contracts are presented in the appendix 1. The instructions are given in terminal session. The instructions are provided by the FabCar samples [Hyperledger fabric 2019. Tutorials]. The SDK node.js test scripts query.js and invoke.js (appendix 2) were used for testing. The code snippets for the FabCar smart contracts and the test report for verifying the FabCar network were not included in this thesis.

6.5 The smart contracts for the PoC were created in the FabCar chaincode file

The smart contracts for the PoC in this thesis were written in JavaScript. The implementation of new smart contracts was done by adding the new functions into the FabCar chaincode file fabcar.js. The smart contracts for FabCar application were left untouched until the development was ready so that they could be used to verify the functionality of smart contracts in general if there were problems with new functions.

One of the new smart contracts named initLedger initializes the blockchain world state with information for ten football players. The script building the network calls the initialization smart contract.

The data records for a football player contain the following fields:

- player Identifier (playerID, indexed by)
- current team
- current football association
- new team
- new football association
- transfer status

There are seven smart contracts in the PoC (appendix 3):

- network initialize with ten football players
- query for all players
- query for one player
- creation of a new player record
- request for the player transfer
- verification for the player transfer
- acceptance for the player transfer

6.6 The FabCar network was rebuild

The network for the PoC was rebuild with the same building script as the original FabCar network was built (appendix 1). The script installs and instantiates the JavaScript chaincode file named fabcar.js containing now the smart contracts for the PoC along the

FabCar smart contracts. The script also initializes the football player objects in the network by creating objects for ten football players.

6.7 The PoC client application was tested with SDK node.js scripts

The PoC client application SDK node.js test scripts were created by using FabCar test scripts as a template. The PoC test template files were named query.js for the queries and invoke.js for updates (appendix 4). The test scripts were run on a terminal session. Different smart contract calls were commented in both files.

The test report was not included in this thesis because it is essentially the same as the test report for API Server PoC test report in Appendix 7.

6.8 The API Server (Express.js) was installed

In the section 6.7 the PoC smart contracts were tested with the scripts where the function calls and parameters were "hard coded" inside the scripts. In the real world the client application should be designed to be more flexible so that the arguments can be specified during executing the client application.

The standard way to query and update a network is to build an interface – called Application Programming Interface (API) – which provides the services for the application developers to access the network.

KC Tam [KC Tam 2019: 24-34] outlines the way how to test the FabCar application with the Express.js API Server. His work was used to give guidelines for building API Server client application services for testing the PoC application. Express.js is an open-source framework for building light web applications and application interfaces.

The instructions to install the Express.js API Server and to rebuild the first-network for the PoC are attached to appendix 5.

6.9 The PoC client application was tested with the API Server scripts

Creating the API Server services for the PoC.

The next phase was to create the API services for the PoC. A file named apiserver.js was created and placed to the directory /apiserver. The API services were written in JavaScript and placed into that file. The PoC was tested by using the same user and peer in all test cases.

The API Server services (Appendix 6) are the following (Figure 29):

1. Query information for all player: **queryallplayers**
2. Query information for one player: **queryplayer**
3. Add information for new player to network: **create_player**
4. Request transfer for a player: **request_transfer**
5. Verify transfer for a player: **verify_transfer**
6. Accept transfer for a player: **accept_transfer**

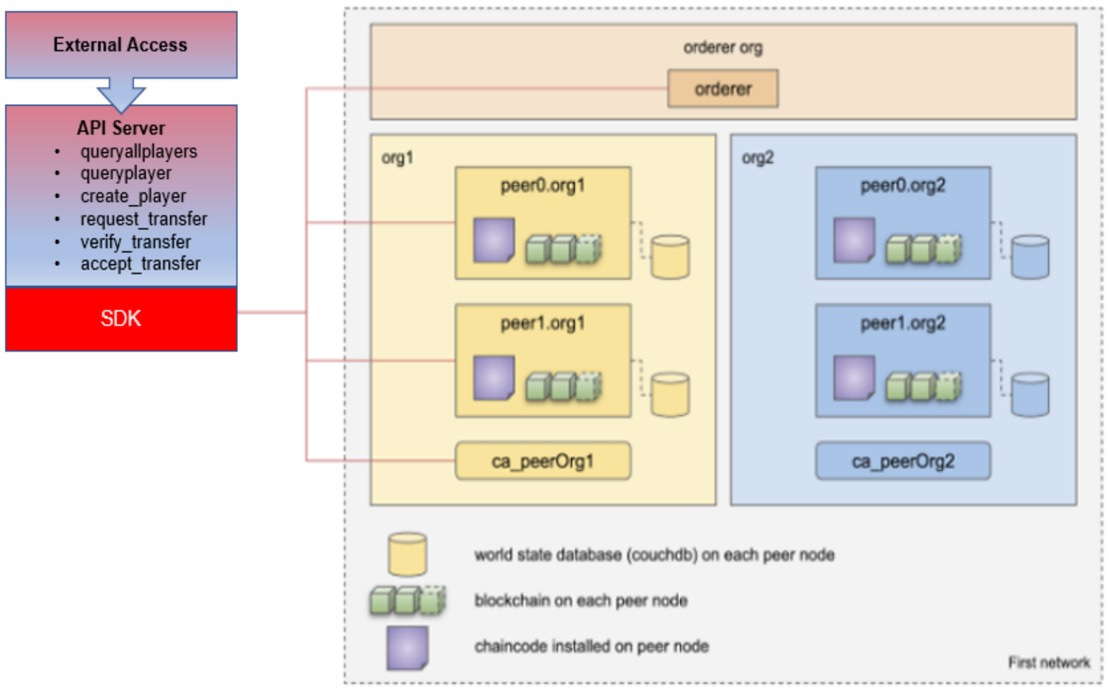


Image (modified): [KC Tam 2019: 24]

Figure 29. The API Server services for the PoC

The API Server file apiserver.js is now ready and the first-network for PoC rebuilt.

Testing the API Server services for the PoC.

The assumption is that the current directory is still /apiserver which was created when installing the API Server.

The API Server services for the PoC are tested via two terminal sessions.

1. The first terminal is opened and the apiserver is started with command ‘node apiserver.js’.
2. The second terminal is opened in order to simulate client application function calls. The testing is done with curl command line tool for making function calls over the network (localhost).

The test cases in the test set are the following:

1. Query information for all players
2. Query information for a given player (id: PLAYER4)
3. Request the transfer for a given player (id: PLAYER4)
4. Query information for a given player (id: PLAYER4) to verify the transfer request
5. Verify the transfer for a given player (id: PLAYER4)
6. Query information for a given player (id: PLAYER4) to verify the transfer verification
7. Accept the transfer for a given player (id: PLAYER4)
8. Query information for a given player (id: PLAYER4) to verify the transfer acceptance
9. Query information for a given player (id: PLAYER10) in order to verify that the player does not exist in the network
10. Add a new player (id: PLAYER10)
11. Query information for the added (id: PLAYER10) to verify that the player was added

The test report containing also the test commands is attached to appendix 7.

7 Summary and Conclusions

The purpose of this thesis was to learn the ways to mitigate the efforts to learn the enterprise blockchain technology and get started with the application development. The requirement was that all the tools are open-source so that they are available for free and that the development environment can be built with a standard personal desktop or laptop.

The international studies showed that while investments on the enterprise blockchain technology are predicted to raise strongly in coming years, the lack of competence, understanding and expertise are becoming an obstacle for the adoption.

It was observed that there is no solid theory for the enterprise blockchain technology. There is a wide selection of enterprise blockchain platforms which features may differ strongly from each other. It was decided to select Hyperledger fabric – one with the biggest market share - to illustrate the features of the enterprise blockchain technology. Hyperledger fabric was also chosen as a platform for the Proof-of-Concept (PoC) application.

As a result of this final year project a PoC application with the open-source tools was built in such a way that it mitigates many of the technical difficulties regarding the development work. The PoC was built by first taking a sample Hyperledger fabric end-to-end blockchain application provided by the Linux Foundation as a basis for the PoC and after that, modifying its smart contracts and the client application for PoC needs. The API Server client application portion was added to the application.

This thesis managed to show that while understanding the enterprise blockchain technology, building a development environment and developing a client application are quite complex tasks, it is possible to find ways to mitigate the learning efforts.

This study shows that while building an end-to-end client application and the network on the Hyperledger fabric platform requires a wide area of competencies and deep understanding of the underlying technology concept, it is possible to build a fully functioning client application with much less effort and competencies but still learn from the process. It is encouraging to be able to build a functioning application and see it working; it is then easy to gradually add features to the network and the client application. The experiences gathered during this project strongly recommend trying this approach if starting to build the application from basics feels discouraging or if the goal is to get the first end-to-end application running as soon as possible.

During the search for information for Hyperledger fabric, it became clear that the literature older than the beginning of year 2019 is mostly obsolete. One reason is that technology is young and immature but maybe the bigger reason is that the support for application development framework Hyperledger Composer was terminated in spring 2019 when the version 1.4 for Hyperledger fabric was published. Quite a large portion of the enterprise blockchain technology books use Hyperledger Composer as an application development framework to illustrate how the application development work is done on Hyperledger fabric platform.

The PoC taught that so far, the application development with Hyperledger fabric platform with open source tools is quite rudimentary scripting work. There are no user-friendly and easy-to-use open source graphical development tools which would improve the adoption of the technology compared to the development tools the Ethereum developer community provides. The reason obviously is that the technology is still so young.

It became obvious during the process that hands-on practical development work courses are needed to enlarge the adoption of the enterprise blockchain technology. The courses found regarding the enterprise blockchain technology were strongly theoretically oriented. Based on the experience the recommendation is that it would speed up the adoption if open Universities and other schools would offer practical hands-on courses on blockchain technologies covering public and enterprise blockchains.

The Proof-of-Concept end-to-end application made as a result of this study could be further developed to be a more comprehensive template which can be used as a basis in building an application. The next development steps could be, for example, adding a graphical interface to the client application and using several user IDs to connect to blockchain network.

References

1 Buterin Vitalik. 2013. White paper. A NEXT GENERATION SMART CONTRACT & DECENTRALIZED APPLICATION PLATFORM. Online material. <https://whitepaper.io/document/5/ethereum-whitepaper>. 2013. Read 20.02.2020.

2 Chung Grant, Luc, Desrosiers, Gupta Manav, Sutton Andrew, Venkatadri Kaushik, Wong Ontak, Zugic Goran. 2019. Performance Tuning and Scaling Enterprise Blockchain Applications. Online material. < https://www.researchgate.net/publication/338158160_Performance_Tuning_and_Scaling_Enterprise_Blockchain_Applications > 24.12..2019. Read 16.05.2020.

3 Consensys. Ethereum & IPFS APIs. Develop now on Web 3.0. Online material. <https://infura.io/> Read 25.02.2020.

4 Ethereum.org. Ethereum. Online material. < https://ethereum.org/>

5 Ganache. Sweet tools for smart contracts: The Truffle Suite gets developers from idea to dapp as comfortably as possible. Online material. <https://www.trufflesuite.com/ > Truffle Blockchain Group. Read 10.03.2020.

6 Haris Adil. 2019. Blockchain—A Short and Simple Explanation with Pictures. Online material. <https://hackernoon.com/blockchain-a-short-and-simple-explanation-with-pictures-d60d652f207f> 5.1.2019. Read 15.2.2020.

7 Hyperledger Composer. 2019. Readme.md. Online material. <https://github.com/hyperledger/composer/blob/master/README.md> 29.08.2019. Read 05.03.2020.

8 Hyperledger fabric. 2019. Testing Using dev mode. Online material. <https://hyperledger-fabric.readthedocs.io/en/release-1.4/chaincode4ade.html#testing-using-dev-mode> 2019. Read 10.03.2020.

9 Hyperledger fabric. 2019. Introduction. Online material. <https://hyperledger-fabric.readthedocs.io/en/release-1.4/whatis.html#hyperledger-fabric> 2019. Read 15.03.2020.

10 Hyperledger fabric. 2019. Blockchain network. Online material. <https://hyperledger-fabric.readthedocs.io/en/release-1.4/network/network.html#blockchain-network> 2019. Read 10.03.2020.

11 Hyperledger fabric. 2019. Ledger. Online material. <https://hyperledger-fabric.readthedocs.io/en/release-1.4/ledger/ledger.html#ledger> 2019. Read 13.02.2020.

12 Hyperledger fabric. 2019. Tutorials. Online material. <https://hyperledger-fabric.readthedocs.io/en/release-1.2/tutorials.html> 2019. Read 12.02.2020.

13 Hyperledger fabric. 2020. Introduction. Online material. <https://hyperledger-fabric.readthedocs.io/en/release-2.0/whatis.html> 01.2020. Read 05.03.2020.

14 KC Tam. 2019. Deep-Dive into Fabcar (revised). Online material. <https://medium.com/@kctheservant/deep-dive-into-fabcar-revised-57cb54642572> 29.08.2019. Read 26.02.2020.

15 Lisi Andrea. 2019. Ethereum DApp development With JavaScript (2019). Online material. <https://elearning.di.unipi.it/pluginfile.php/25500/mod_resource/content/1/22-05-19-DApp%20Development.pdf> 2019. Read 11.2.2020.

16 McCubbin Gregory. 2020. Become an In-demand Blockchain Master. Online material. <https://www.dappuniversity.com/>. Dapp University. 2019. Read 10.02.2020.

17 MetaMask. A crypto wallet & gateway to blockchain apps. Online material. <https://metamask.io/> A ConsenSys Formation. Read 15.02.2020.

18 Pawczuk Linda, Massey Rob, Holdowsky Jonathan. 2019. Deloitte's 2019 Global Blockchain Survey. Blockchain gets down to business. Online material. <https://www2.deloitte.com/us/en/insights/topics/understanding-blockchain-potential/global-blockchain-survey.html> London: Deloitte Touche Tohmatsu Limited. 06.05.2019. Read 25.03.2020.

19 Rauchs Michel, Blandin Apolline, Bear Keith, McKeon Stephen. 2019. 2nd Global Enterprise Blockchain Benchmarking Study. Online material. <https://www.jbs.cam.ac.uk/fileadmin/user_upload/research/centres/alternative-finance/downloads/2019-ccaf-second-global-enterprise-blockchain-report.pdf> Cambridge: University of Cambridge. 18.09.2019. Read 20.03.2020.

20 Satoshi Nakamoto. 2008. White paper. Bitcoin: A Peer-to-Peer Electronic Cash System. Online material. <https://bitcoin.org/bitcoin.pdf> Read 18.02.2020.

21 Singhal Bikramaditya, Dhameja Gautam, Panda Priyansu Sekhar. 2018. Beginning Blockchain. A Beginner's Guide to Building Blockchain Solutions. New York: Apress.

22 Truffle Suite. Sweet tools for smart contracts: The Truffle Suite gets developers from idea to dapp as comfortably as possible. Online material. <https://www.truf-flesuite.com/> Truffle Blockchain Group. Read 05.02.2020.

23 Warren Sheila, Treat David. 2019. Building Value with Blockchain Technology: Is Blockchain Worth the Investment? Online material. <https://www.accen-ture.com/_acnmedia/pdf-105/accenture-blockchain-value-report.pdf> Geneva: World Economic Forum. 05.2019. Read 25.03.2020.

24 Welfare Antony. 2019. Commercializing Blockchain. Strategic Applications in the Real World. United Kingdom: Wiley.

25 Wu Xun, Zou Zhihong, Song Dongying. 2019. Learn Ethereum Online material. <https://learning.oreilly.com/library/view/learn-ethereum/9781789954111/> Bir-mingham: Packt Publishing. 09.2019. Read 15.05.2020.

The instructions for building the FabCar network and its smart contracts

Hyperledger fabric provides [Hyperledger fabric 2019: Tutorials] a script that downloads and installs samples and binaries which can be used as a basis for learning and application development.

1. Downloading the FabCar Samples, Binaries and Docker Images

First phase is to create a working directory where the samples will be downloaded. After moving to the working directory, the following commands download the samples (Hyperledger fabric version 1.4.3) to the working directory and creates subdirectory named "fabric-samples":

- curl -sSL http://bit.ly/2ysbOFE | bash -s -- 1.4.3
- export PATH=<path to working directory location>/bin:$PATH

2. Building the network and deploying the smart contracts functions for FabCar

The assumption is that the current location now is the working directory which was created in the previous section.

The commands in this sections are used to create fabcar network where there are two organisations, two peers, one SOLO orderer, Certificate Authority, one channel and fabcar chaincode functions (smart contracts).

The following set of commands builds the fabcar network and its chaincode functions:

1. cd fabric-samples/first-network
2. ./byfn.sh down
3. docker rm -f $(docker ps -aq)
4. docker rmi -f $(docker images | grep fabcar | awk '{print $3}')
5. cd ../fabcar
6. ./startFabric.sh javascript
7. cd javascript

8. npm install
9. rm -rf wallet
10. node enrollAdmin.js
11. node registerUser.js

SDK Node.js programs for querying and updating FabCar network

1. SDK node.js query template for testing the FabCar sample application queries

```
'use strict';
const { FileSystemWallet, Gateway } = require('fabric-network');
const path = require('path');
const ccpPath = path.resolve(__dirname, '..', '..', 'first-network', 'connection-org1.json');

async function main() {
    try {
        // Create a new file system based wallet for managing identities.
        const walletPath = path.join(process.cwd(), 'wallet');
        const wallet = new FileSystemWallet(walletPath);
        console.log(`Wallet path: ${walletPath}`);

        // Check to see if we've already enrolled the user.
        const userExists = await wallet.exists('user1');
        if (!userExists) {
            console.log('An identity for the user "user1" does not exist in the wallet');
            console.log('Run the registerUser.js application before retrying');
            return;
        }

        // Create a new gateway for connecting to our peer node.
        const gateway = new Gateway();
        await gateway.connect(ccpPath, { wallet, identity: 'user1', discovery: { enabled: true, asLocalhost: true } });

        // Get the network (channel) our contract is deployed to.
        const network = await gateway.getNetwork('mychannel');

        // Get the contract from the network.
        const contract = network.getContract('fabcar');

        // Evaluate the specified transaction.
        // queryCar transaction - requires 1 argument, ex: ('queryCar', 'CAR4')
        // queryAllCars transaction - requires no arguments, ex: ('queryAllCars')
        const result = await contract.evaluateTransaction('queryAllCars');
        console.log(`Transaction has been evaluated, result is: ${result.toString()}`);

    } catch (error) {
        console.error(`Failed to evaluate transaction: ${error}`);
        process.exit(1);
    }
}
main();
```

2. SDK node.js update template for testing the FabCar sample application updates

```
'use strict';
const { FileSystemWallet, Gateway } = require('fabric-network');
const path = require('path');
const ccpPath = path.resolve(__dirname, '..', '..', 'first-network', 'connection-org1.json');

async function main() {
    try {
        // Create a new file system based wallet for managing identities.
        const walletPath = path.join(process.cwd(), 'wallet');
        const wallet = new FileSystemWallet(walletPath);
        console.log(`Wallet path: ${walletPath}`);

        // Check to see if we've already enrolled the user.
        const userExists = await wallet.exists('user1');
        if (!userExists) {
            console.log('An identity for the user "user1" does not exist in the wallet');
            console.log('Run the registerUser.js application before retrying');
            return;
        }

        // Create a new gateway for connecting to our peer node.
        const gateway = new Gateway();
        await gateway.connect(ccpPath, { wallet, identity: 'user1', discovery: { enabled: true, asLocalhost: true } });

        // Get the network (channel) our contract is deployed to.
        const network = await gateway.getNetwork('mychannel');

        // Get the contract from the network.
        const contract = network.getContract('fabcar');

        // Submit the specified transaction.
        // createCar transaction - requires 5 argument, ex: ('createCar', 'CAR12', 'Honda', 'Accord', 'Black', 'Tom')
        // changeCarOwner transaction - requires 2 args , ex: ('changeCarOwner', 'CAR10', 'Dave')
        await contract.submitTransaction('createCar', 'CAR12', 'Honda', 'Accord', 'Black', 'Tom');
        console.log('Transaction has been submitted');

        // Disconnect from the gateway.
        await gateway.disconnect();

    } catch (error) {
        console.error(`Failed to submit transaction: ${error}`);
        process.exit(1);
    }
}
main();
```

The code snippets for the Proof-of-Concept smart contracts

1. The structure of the chaincode file containing the smart contracts

```
 // Smart contracts for football player registrations
'use strict';
const { Contract } = require('fabric-contract-api');

class FabCar extends Contract {

    // Network is initialized with ten players
    async initLedger(ctx) {
            // The body of the function to be added here (attached below)
    }

    // Returns player info for the given player
    async queryPlayer(ctx, playerNumber) {
            // The body of the function to be added here (attached below)
    }

    // Function creates a new player
    async createPlayer(ctx, playerNumber, playerName, currentTeam, currentAssociation) {
            // The body of the function to be added here (attached below)
    }

    // Function returns infomation for all players - assumption is that there is max 1000 players
    async queryAllPlayers(ctx) {
            // The body of the function to be added here (attached below)
    }

    // Player has reguested for the transfer. New Team updates the status as "Requested"
    async requestTransfer(ctx, playerNumber, newTeam, newAssociation) {
            // The body of the function to be added here (attached below)
    }

    // Current Team verifies that transfer is ok and updates the status as "Verified"
    async verifyTransfer(ctx, playerNumber) {
            // The body of the function to be added here (attached below)
    }

    // Football association accepts the transfer and updates the status as "Accepted"
    async acceptTransfer(ctx, playerNumber) {
            // The body of the function to be added here (attached below)
    }
}
module.exports = FabCar;
```

2. initLedger – Smart contract initializes the network initialization ten players

```
// Network is initialized with ten players
async initLedger(ctx) {
    console.info('============= START : Initialize Ledger ====================');
    const players = [{
            playerName: 'Pele',
            currentTeam: 'Rops',
            currentAssociation: 'Football Association of Finland',
            newTeam: ' ',
            newAssociation: ' ',
            transferStatus: 'Accepted',
        },
        {
            playerName: 'Ronaldo',
            currentTeam: 'Sao Paulo',
            currentAssociation: 'Brazilian Football Confederation',
            newTeam: ' ',
            newAssociation: ' ',
            transferStatus: 'Accepted',
        },
        ……      Note: ”Seven of initialization records are left away from this
        ……                  list compared to the original snippet”
        {
            playerName: 'Tolsa',
            currentTeam: 'KTP',
            currentAssociation: 'Football Association of Finland',
            newTeam: ' ',
            newAssociation: ' ',
            transferStatus: 'Accepted',
        },
    ];
    for (let i = 0; i < players.length; i++) {
        players[i].docType = 'player';
        await ctx.stub.putState('PLAYER' + i, Buffer.from(JSON.stringify(players[i])));
        console.info('Added <--> ', players[i]);
    }
    console.info('============= END : Initialize Ledger ==================');
```

3. queryAllPlayers - Smart contract returns information for all players

```
// Function returns infomation for all players - max 1000 players

    async queryAllPlayers(ctx) {
        const startKey = 'PLAYER0';
        const endKey = 'PLAYER999';

        const iterator = await ctx.stub.getStateByRange(startKey, endKey);
        const allResults = [];
        while (true) {
            const res = await iterator.next();

            if (res.value && res.value.value.toString()) {
                console.log(res.value.value.toString('utf8'));

                const Key = res.value.key;
                let Record;
                try {
                    Record = JSON.parse(res.value.value.toString('utf8'));
                } catch (err) {
                    console.log(err);
                    Record = res.value.value.toString('utf8');
                }
                allResults.push({ Key, Record });
            }
            if (res.done) {
                console.log('end of data');
                await iterator.close();
                console.info(allResults);
                return JSON.stringify(allResults);
            }
        }
    }
```

4. queryPlayer - Smart contract returns information for a given player

```
    // Function returns player info for a given player
    async queryPlayer(ctx, playerNumber) {
        // get the player from chaincode state
        const playerAsBytes = await ctx.stub.getState(playerNumber);
        if (!playerAsBytes || playerAsBytes.length === 0) {
            throw new Error(`${playerNumber} does not exist`);
        }
        console.log(playerAsBytes.toString());
        return playerAsBytes.toString();
    }
```

5. createPlayer - Smart contract adds a new player to network

```
// Funtion creates a new player
async createPlayer(ctx, playerNumber, playerName, currentTeam, currentAssociation) {

    const player = {
        playerName,
        docType: 'player',
        currentTeam,
        currentAssociation,
        newTeam: ' ',
        newAssociation: ' ',
        transferStatus: 'Accepted',
    };

    await ctx.stub.putState(playerNumber, Buffer.from(JSON.stringify(player)));
}
```

6. requestTransfer - Smart contract updates the transfer status as “Requested”

```
// Player has reguested for the transfer. New Team updates status as "Requested"
async requestTransfer(ctx, playerNumber, newTeam, newAssociation) {
    console.info('============= START : requestTransfer ===========');
    // get the player from chaincode state
    const playerAsBytes = await ctx.stub.getState(playerNumber);
    if (!playerAsBytes || playerAsBytes.length === 0) {
        throw new Error(`${playerNumber} does not exist`);
    }
    const player = JSON.parse(playerAsBytes.toString());

    player.newTeam = newTeam;
    player.newAssociation = newAssociation;
    player.transferStatus = 'Requested';

    await ctx.stub.putState(playerNumber, Buffer.from(JSON.stringify(player)));
    console.info('============= END : reaquestTransfer ===========');
}
```

8. verifyTransfer - Smart contract updates the transfer status as “Verified”

```
// Current Team verifies that transfer is ok and updates the status as "Verified"
async verifyTransfer(ctx, playerNumber) {
    console.info('============= START : verifyTransfer ===========');
    // get the player from chaincode state
    const playerAsBytes = await ctx.stub.getState(playerNumber);
    if (!playerAsBytes || playerAsBytes.length === 0) {
        throw new Error(`${playerNumber} does not exist`);
    }
    const player = JSON.parse(playerAsBytes.toString());

    player.transferStatus = 'Verified';

    await ctx.stub.putState(playerNumber, Buffer.from(JSON.stringify(player)));
    console.info('============= END : verifyTransfer ===========');
}
```

9. acceptTransfer - Function updates the transfer status as “Accepted”

```
// Football association accepts the transfer and updates the status as "Accepted"
async acceptTransfer(ctx, playerNumber) {
    // get the player from chaincode state
    const playerAsBytes = await ctx.stub.getState(playerNumber);
    if (!playerAsBytes || playerAsBytes.length === 0) {
        throw new Error(`${playerNumber} does not exist`);
    }
    const player = JSON.parse(playerAsBytes.toString());

    player.currentTeam = player.newTeam;
    player.currentAssociation = player.newAssociation;
    player.newTeam = ' ';
    player.newAssociation = ' ';
    player.transferStatus = 'Accepted';

    await ctx.stub.putState(playerNumber, Buffer.from(JSON.stringify(player)));
}
```

SDK Node.js programs for querying and updating PoC network

1. SDK node.js query template for testing the PoC client application queries

```
// Filename is Query.js
// It is the same file as in Appendix 2, page 1
// Except the portion of the code below was modified

    // Evaluate the specified transaction.
    // queryPlayer transaction - requires 1 argument, ex: ('queryPlayer', 'PLAYER4')
    // queryAllPlayers transaction - requires no arguments, ex: ('queryAllPlayers')
    const result = await contract.evaluateTransaction('queryAllPlayers');
    console.log(`Transaction has been evaluated, result is: ${result.toString()}`);
```

2. SDK node.js update template for testing the PoC client application updates

```
// Filename is Invoke.js
// It is the same file as in Appendix 2, page 2
// Except the portion of the code below was modified

    // Submit the specified transaction.
    // requestTransfer transaction - requires 3 arguments, ex: ('requestTransfer', 'PLAYER4', 'Rops',
                                                    'Football Association of Finland')
    // verifyTransfer transaction - requires 1 arg , ex: ('verifyTransfer', 'PLAYER4')
    // acceptTransfer transaction - requires 1 arg , ex: ('acceptTransfer', 'PLAYER4')
    // createPlayer transaction - requires 4 arguments, ex: ('createPlayer', 'PLAYER10', 'Baresi',
                                                  'AC Milano',  'Italian Football Federation')
    await contract.submitTransaction('createPlayer', 'PLAYER10', 'Baresi', 'AC Milano', 'Italian
                                                  Football Federation');
```

Installing API Server Express.js and rebuilding the PoC network

This section contains the instructions to install Express.js as a node.js API Server and to rebuild the FabCar first-network containing the smart contracts for PoC.

The assumption in this section is that the PoC network and functions are already earlier created and tested working.

In order to avoid unintentionally breaking the working Proof-of-Concept application Express.js API Server was installed to a separate directory named /fabric-samples/fabcar/apiserver. The directory was created by replicating /javascript directory to a directory named /apiserver.

The commands for installing API Server Express.js are the following:

1. cd fabric-samples/fabcar/
2. cp -r javascript/ apiserver/
3. cd apiserver
4. npm install express body-parser –save

The commands for rebuilding the first-network for PoC from the /apiserver directory:

5. cd ../../first-network
6. ./byfn.sh down
7. docker rm -f $(docker ps -aq)
8. docker rmi -f $(docker images | grep fabcar | awk '{print $3}')
9. cd ../fabcar
10. ./startFabric.sh javascript
11. cd apiserver
12. npm install
13. rm -rf wallet
14. node enrollAdmin.js
15. node registerUser.js

Creating the API Server services for the PoC.

Next phase was to create the API services for the PoC. A file named apiserver.js was created and placed to the directory /apiserver. The API services were written in JavaS-cript and placed into that file. The work in this section was inspired by the KC Tam article [KC Tam 2019: 24-34].

The API services created in the file apiserver.js (Appendix 6) are the following:

7. Query information for all player: **queryallplayers.**
8. Query information for one player: **queryplayer.**
9. Add information for new player to network: **create_player.**
10. Request transfer for a player: **request_transfer.**
11. Verify transfer for a player: **verify_transfer.**
12. Accept transfer for a player: **accept_transfer.**

The API Server file apiserver.js is now ready and the first-network for PoC rebuilt.

Testing the API Server services made for the PoC.

The assumption is that the current directory is still /apiserver which was created when installing the API Server.

The API Server services for the PoC are tested via two terminal sessions.

3. The first terminal is opened and the apiserver is started with command ‘node apiserver.js’.
4. The second terminal is opened in order to simulate client application function calls. The testing is done with curl command line tool for making function calls over the network (localhost). The test report is attached in appendix 7.

API server test scripts for testing the Proof-of-Concept (PoC)

1. Structure of the API Server script file named apiserver.js for PoC tests

```
/* API server script for the football player registration application */

'use strict';
const express = require('express');
const bodyParser = require('body-parser');
const app = express();
app.use(bodyParser.json());

// Setting for Hyperledger Fabric
const { FileSystemWallet, Gateway } = require('fabric-network');
const path = require('path');
const ccpPath = path.resolve(__dirname, '..', '..', 'first-network', 'connection-org1.json');

app.get('/api/queryallplayers', async function(req, res) {
    // Function returns information for all players
    // Body for the function to be added here (attached below)
});
app.get('/api/queryplayer/:player_index', async function(req, res) {
    // Function returns information for one given player
    // Body for the function to be added here (attached below)
});
app.post('/api/create_player/', async function(req, res) {
    // Function invokes the smart contract to create a new player
    // Body for the function to be added here (attached below)
});
app.put('/api/request_transfer/:player_index', async function(req, res) {
    // Function invokes the smart contract to change the transfer status to "Requested"
    // and adds the information for the new team to player record
    // Body for the function to be added here (attached below)
});
app.put('/api/verify_transfer/:player_index', async function(req, res) {
    // Function invokes the smart contract to change the transfer status to "Verified"
    // Body for the function to be added here (attached below)
});
app.put('/api/accept_transfer/:player_index', async function(req, res) {
    // Function invokes the smart contract to change the transfer status to "Accepted"
    // and updates the current team information with new team information
    // Body for the function to be added here (attached below)
});

app.listen(8080);
```

2. queryallplayers: Function returns the information for all players – no args

```
app.get('/api/queryallplayers', async function(req, res) {
    // Function returns information for all players
    try {

        // Create a new file system based wallet for managing identities.
        const walletPath = path.join(process.cwd(), 'wallet');
        const wallet = new FileSystemWallet(walletPath);
        console.log(`Wallet path: ${walletPath}`);

        // Check to see if we've already enrolled the user.
        const userExists = await wallet.exists('user1');
        if (!userExists) {
            console.log('An identity for the user "user1" does not exist in the wallet');
            console.log('Run the registerUser.js application before retrying');
            return;
        }

        // Create a new gateway for connecting to our peer node.
        const gateway = new Gateway();
        await gateway.connect(ccpPath, { wallet, identity: 'user1', discovery: { enabled: true, asLocalhost: true } });

        // Get the network (channel) our contract is deployed to.
        const network = await gateway.getNetwork('mychannel');

        // Get the contract from the network.
        const contract = network.getContract('fabcar');

        // Evaluate the specified transaction.
        // queryAllPlayers transaction - requires no arguments, ex: ('queryAllPlayers')
        const result = await contract.evaluateTransaction('queryAllPlayers');
        console.log(`Transaction has been evaluated, result is: ${result.toString()}`);
        res.status(200).json({ response: result.toString() });

    } catch (error) {
        console.error(`Failed to evaluate transaction: ${error}`);
        res.status(500).json({ error: error });
        process.exit(1);
    }
});
```

3. queryplayer: Function returns the information for a given player – 1 args

```
// The repeated code from the file in the Appendix 6, page 2 was removed from the code below
// and replaced by character string ”…………”

app.get('/api/queryplayer/:player_index', async function(req, res) {
    // Function returns information for one given player
   ………..
        // Evaluate the specified transaction.
        // queryPlayer transaction - requires 1 argument, ex: ('queryPlayer', 'CAR4')
        const result = await contract.evaluateTransaction('queryPlayer', req.params.player_index);
   ……….
});
```

4. create_player: Function to invoke creation of a new player – 3 args

```
// The repeated code from the file in the Appendix 6, page 2 was removed from the code below
// and replaced by character string ”…………”

app.post('/api/create_player/', async function(req, res) {
    // Function invokes the smart contract to create a new player
        ……………………..

        // Submit the specified transaction.
        // createPlayer transaction - requires 4 arguments, ex: ('createPlayer', 'PLAYER10', 'Baresi',
                                                        'AC Milano', 'Italian Football Federation')
        await contract.submitTransaction('createPlayer', req.body.player_number,
                req.body.player_name, req.body.current_team, req.body.current_association);
        ……………………..
});
```

5. request_transfer: Function to change status to “Requested” - 3 args

```
// The repeated code from the file in the Appendix 6, page 2 was removed from the code below
// and replaced by character string ”…………”

app.put('/api/request_transfer/:player_index', async function(req, res) {
    // Function invokes the smart contract to change the transfer status to "Requested"
    // and adds the information for the new team to player record

    ………………….

        // Submit the specified transaction.
        // requestTransfer transaction - requires 3 arguments, ex: ('requestTransfer', 'PLAYER4',
                                                                   'Rops', 'Football Association of Finland')
        await contract.submitTransaction('requestTransfer', req.params.player_index,
                req.body.new_team, req.body.new_association);
     ……………….
});
```

6. verify_transfer: Function to change transfer status to “Verified” - 1 args

```
// The repeated code from the file in the Appendix 6, page 2 was removed from the code below
// and replaced by character string ”…………”

app.put('/api/verify_transfer/:player_index', async function(req, res) {
    // Function invokes the smart contract to change the transfer status to "Verified"
    ………………..

        // Submit the specified transaction.
        // verifyTransfer transaction - requires 1 arg , ex: ('verifyTransfer', 'PLAYER4')
        await contract.submitTransaction('verifyTransfer', req.params.player_index);
    ………………
});
```

7. accept_transfer: Function to change transfer status to “Accepted” - 1 args

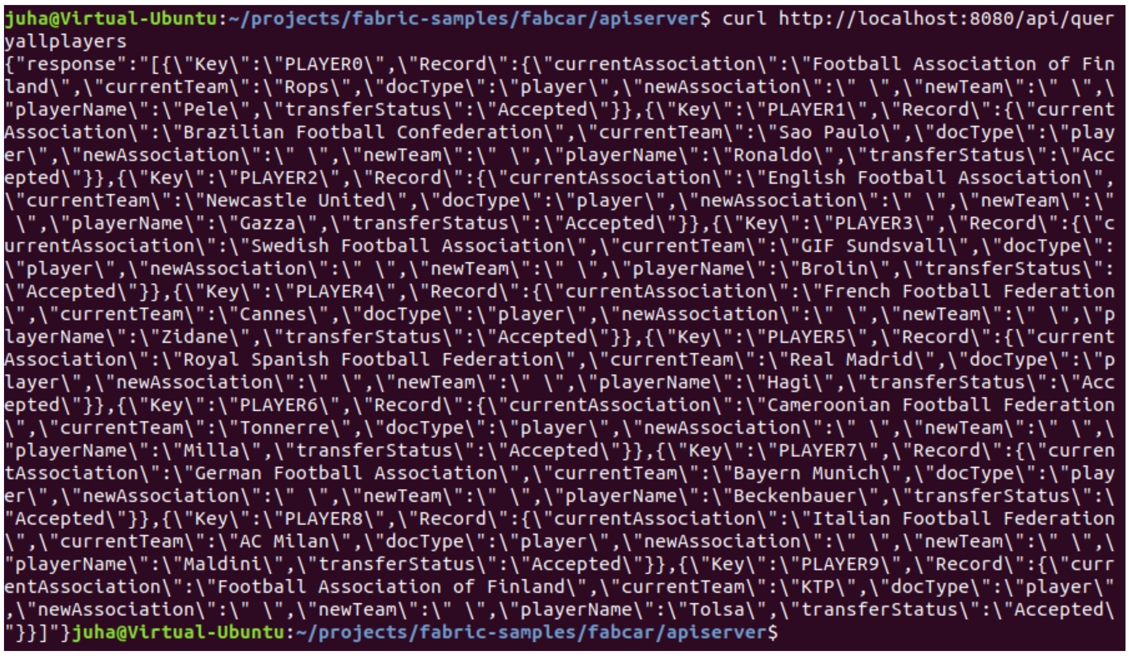
```
// The repeated code from the file in the Appendix 6, page 2 was removed from the code below
// and replaced by character string ”…………”

app.put('/api/accept_transfer/:player_index', async function(req, res) {
    // Function invokes the smart contract to change the transfer status to "Accepted"
    // and updates the current team information with new team information
    ……………….

        // Submit the specified transaction.
        // acceptTransfer transaction - requires 1 arg , ex: ('acceptTransfer', 'PLAYER4')
        await contract.submitTransaction('acceptTransfer', req.params.player_index);
        console.log('Transaction has been submitted');
    ……………….
});
```

API Server test report for the Proof-of-Concept testing

1. Function for returning the information for all players

juha@Virtual-Ubuntu:~/projects/fabric-samples/fabcar/apiserver$ **curl http://localhost:8080/api/queryallplayers**

```
{"response":"[
{\"Key\":\"PLAYER0\",
   \"Record\":{
                \"currentAssociation\":\"Football Association of Finland\",
                \"currentTeam\":\"Rops\",
                \"docType\":\"player\",
                \"newAssociation\":\" \",
                \"newTeam\":\" \",
                \"playerName\":\"Pele\",
                \"transferStatus\":\"Accepted\"}},
{\"Key\":\"PLAYER1\",
   \"Record\":{
                \"currentAssociation\":\"Brazilian Football Confederation\",
                \"currentTeam\":\"Sao Paulo\",
                \"docType\":\"player\",
                \"newAssociation\":\" \",
                \"newTeam\":\" \",
                \"playerName\":\"Ronaldo\",
                \"transferStatus\":\"Accepted\"}},

{\"Key\":\"PLAYER2\",
   \"Record\":{
                \"currentAssociation\":\"English Football Association\",
                \"currentTeam\":\"Newcastle United\",
                \"docType\":\"player\",
                \"newAssociation\":\" \",
                \"newTeam\":\" \",
                \"playerName\":\"Gazza\",
                \"transferStatus\":\"Accepted\"}},

{\"Key\":\"PLAYER3\",
   \"Record\":{
                \"currentAssociation\":\"Swedish Football Association\",
                \"currentTeam\":\"GIF Sundsvall\",
                \"docType\":\"player\",
                \"newAssociation\":\" \",
                \"newTeam\":\" \",
                \"playerName\":\"Brolin\",
                \"transferStatus\":\"Accepted\"}},

{\"Key\":\"PLAYER4\",
   \"Record\":{
                \"currentAssociation\":\"French Football Federation\",
                \"currentTeam\":\"Cannes\",
                \"docType\":\"player\",
                \"newAssociation\":\" \",
                \"newTeam\":\" \",
                \"playerName\":\"Zidane\",
                \"transferStatus\":\"Accepted\"}},

{\"Key\":\"PLAYER5\",
   \"Record\":{
                    \"currentAssociation\":\"Royal Spanish Football Federation\",
                    \"currentTeam\":\"Real Madrid\",
                    \"docType\":\"player\",
                    \"newAssociation\":\" \",
                    \"newTeam\":\" \",
                    \"playerName\":\"Hagi\",
                    \"transferStatus\":\"Accepted\"}},

{\"Key\":\"PLAYER6\",
   \"Record\":{
                    \"currentAssociation\":\"Cameroonian Football Federation\",
                    \"currentTeam\":\"Tonnerre\",
                    \"docType\":\"player\",
                    \"newAssociation\":\" \",
                    \"newTeam\":\" \",
                    \"playerName\":\"Milla\",
                    \"transferStatus\":\"Accepted\"}},

{\"Key\":\"PLAYER7\",
   \"Record\":{
                    \"currentAssociation\":\"German Football Association\",
                    \"currentTeam\":\"Bayern Munich\",
                    \"docType\":\"player\",
                    \"newAssociation\":\" \",
                    \"newTeam\":\" \",
                    \"playerName\":\"Beckenbauer\",
                    \"transferStatus\":\"Accepted\"}},

{\"Key\":\"PLAYER8\",
   \"Record\":{
                    \"currentAssociation\":\"Italian Football Federation\",
                    \"currentTeam\":\"AC Milan\",
                    \"docType\":\"player\",
                    \"newAssociation\":\" \",
                    \"newTeam\":\" \",
                    \"playerName\":\"Maldini\",
                    \"transferStatus\":\"Accepted\"}},

{\"Key\":\"PLAYER9\",
   \"Record\":{
                    \"currentAssociation\":\"Football Association of Finland\",
                    \"currentTeam\":\"KTP\",
                    \"docType\":\"player\",
                    \"newAssociation\":\" \",
                    \"newTeam\":\" \",
                    \"playerName\":\"Tolsa\",
                    \"transferStatus\":\"Accepted\"}}]"}
```

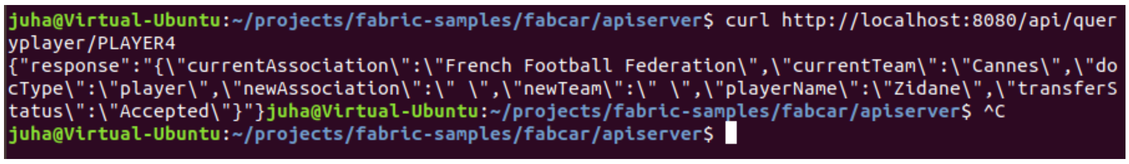


```
juha@Virtual-Ubuntu:~/projects/fabric-samples/fabcar/apiserver$ curl http://localhost:8080/api/quer
yallplayers
{"response":"[{\"Key\":\"PLAYER0\",\"Record\":{\"currentAssociation\":\"Football Association of Fin
land\",\"currentTeam\":\"Rops\",\"docType\":\"player\",\"newAssociation\":\" \",\"newTeam\":\" \",\
"playerName\":\"Pele\",\"transferStatus\":\"Accepted\"}},{\"Key\":\"PLAYER1\",\"Record\":{\"current
Association\":\"Brazilian Football Confederation\",\"currentTeam\":\"Sao Paulo\",\"docType\":\"play
er\",\"newAssociation\":\" \",\"newTeam\":\" \",\"playerName\":\"Ronaldo\",\"transferStatus\":\"Acc
epted\"}},{\"Key\":\"PLAYER2\",\"Record\":{\"currentAssociation\":\"English Football Association\",
\"currentTeam\":\"Newcastle United\",\"docType\":\"player\",\"newAssociation\":\" \",\"newTeam\":\"
 \",\"playerName\":\"Gazza\",\"transferStatus\":\"Accepted\"}},{\"Key\":\"PLAYER3\",\"Record\":{\"c
urrentAssociation\":\"Swedish Football Association\",\"currentTeam\":\"GIF Sundsvall\",\"docType\":
\"player\",\"newAssociation\":\" \",\"newTeam\":\" \",\"playerName\":\"Brolin\",\"transferStatus\":
\"Accepted\"}},{\"Key\":\"PLAYER4\",\"Record\":{\"currentAssociation\":\"French Football Federation
\",\"currentTeam\":\"Cannes\",\"docType\":\"player\",\"newAssociation\":\" \",\"newTeam\":\" \",\"p
layerName\":\"Zidane\",\"transferStatus\":\"Accepted\"}},{\"Key\":\"PLAYER5\",\"Record\":{\"current
Association\":\"Royal Spanish Football Federation\",\"currentTeam\":\"Real Madrid\",\"docType\":\"p
layer\",\"newAssociation\":\" \",\"newTeam\":\" \",\"playerName\":\"Hagi\",\"transferStatus\":\"Acc
epted\"}},{\"Key\":\"PLAYER6\",\"Record\":{\"currentAssociation\":\"Cameroonian Football Federation
\",\"currentTeam\":\"Tonnerre\",\"docType\":\"player\",\"newAssociation\":\" \",\"newTeam\":\" \",\
"playerName\":\"Milla\",\"transferStatus\":\"Accepted\"}},{\"Key\":\"PLAYER7\",\"Record\":{\"curren
tAssociation\":\"German Football Association\",\"currentTeam\":\"Bayern Munich\",\"docType\":\"play
er\",\"newAssociation\":\" \",\"newTeam\":\" \",\"playerName\":\"Beckenbauer\",\"transferStatus\":\
"Accepted\"}},{\"Key\":\"PLAYER8\",\"Record\":{\"currentAssociation\":\"Italian Football Federation
\",\"currentTeam\":\"AC Milan\",\"docType\":\"player\",\"newAssociation\":\" \",\"newTeam\":\" \",\
"playerName\":\"Maldini\",\"transferStatus\":\"Accepted\"}},{\"Key\":\"PLAYER9\",\"Record\":{\"curr
entAssociation\":\"Football Association of Finland\",\"currentTeam\":\"KTP\",\"docType\":\"player\"
,\"newAssociation\":\" \",\"newTeam\":\" \",\"playerName\":\"Tolsa\",\"transferStatus\":\"Accepted\
"}}]"}juha@Virtual-Ubuntu:~/projects/fabric-samples/fabcar/apiserver$
```

2. Function for returning the information for one given player (id: PLAYER4)

juha@Virtual-Ubuntu:~/projects/fabric-samples/fabcar/apiserver$ **curl http://localhost:8080/api/queryplayer/PLAYER4**

```
{"response":"{
              \"currentAssociation\":\"French Football Federation\",
              \"currentTeam\":\"Cannes\",
              \"docType\":\"player\",
              \"newAssociation\":\" \",
              \"newTeam\":\" \",\"playerName\":\"Zidane\",
              \"transferStatus\":\"Accepted\"}"}
```

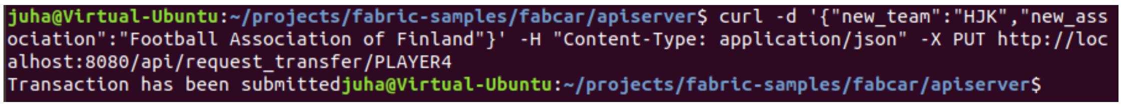


```
juha@Virtual-Ubuntu:~/projects/fabric-samples/fabcar/apiserver$ curl http://localhost:8080/api/quer
yplayer/PLAYER4
{"response":"{\"currentAssociation\":\"French Football Federation\",\"currentTeam\":\"Cannes\",\"do
cType\":\"player\",\"newAssociation\":\" \",\"newTeam\":\" \",\"playerName\":\"Zidane\",\"transferS
tatus\":\"Accepted\"}"}juha@Virtual-Ubuntu:~/projects/fabric-samples/fabcar/apiserver$ ^C
juha@Virtual-Ubuntu:~/projects/fabric-samples/fabcar/apiserver$ 
```

3. Function for changing the transfer status to “Requested” (id: PLAYER4)

juha@Virtual-Ubuntu:~/projects/fabric-samples/fabcar/apiserver$ **curl -d '{"new_team":"HJK","new_association":"Football Association of Finland"}' -H "Content-Type: application/json" -X PUT http://localhost:8080/api/request_transfer/PLAYER4**

Transaction has been submitted



```
juha@Virtual-Ubuntu:~/projects/fabric-samples/fabcar/apiserver$ curl -d '{"new_team":"HJK","new_ass
ociation":"Football Association of Finland"}' -H "Content-Type: application/json" -X PUT http://loc
alhost:8080/api/request_transfer/PLAYER4
Transaction has been submittedjuha@Virtual-Ubuntu:~/projects/fabric-samples/fabcar/apiserver$
```

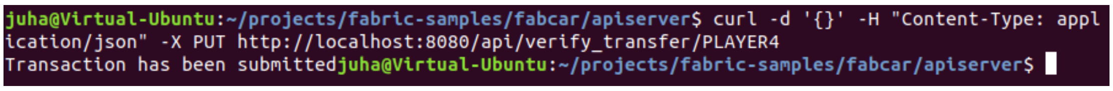
4. Function for returning the information for one given player (id: PLAYER4)

juha@Virtual-Ubuntu:~/projects/fabric-samples/fabcar/apiserver$ **curl http://localhost:8080/api/queryplayer/PLAYER4**

{"response":"{
\"currentAssociation\":\"French Football Federation\",
\"currentTeam\":\"Cannes\",
\"docType\":\"player\",
\"newAssociation\":\"**Football Association of Finland**\",
\"newTeam\":\"**HJK**\",
\"playerName\":\"Zidane\",
\"transferStatus\":\"**Requested**\"}"}

5. Function for changing the transfer status to "Verified" (id: PLAYER4)

juha@Virtual-Ubuntu:~/projects/fabric-samples/fabcar/apiserver$ **curl -d '{}' -H "Content-Type: application/json" -X PUT http://localhost:8080/api/verify_transfer/PLAYER4**
Transaction has been submitted

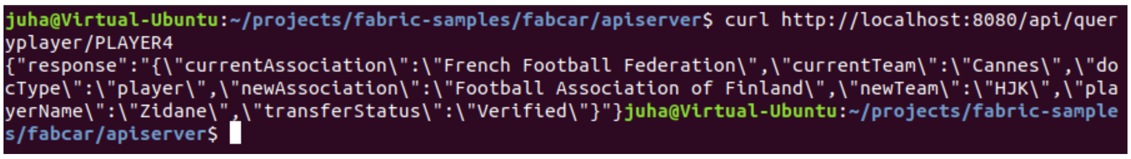


```
juha@Virtual-Ubuntu:~/projects/fabric-samples/fabcar/apiserver$ curl -d '{}' -H "Content-Type: appl
ication/json" -X PUT http://localhost:8080/api/verify_transfer/PLAYER4
Transaction has been submittedjuha@Virtual-Ubuntu:~/projects/fabric-samples/fabcar/apiserver$ 
```

6. Function for returning the information for one given player (id: PLAYER4)

juha@Virtual-Ubuntu:~/projects/fabric-samples/fabcar/apiserver$ **curl http://localhost:8080/api/queryplayer/PLAYER4**

{"response":"{
\"currentAssociation\":\"French Football Federation\",
\"currentTeam\":\"Cannes\",
\"docType\":\"player\",
\"newAssociation\":\"Football Association of Finland\",
\"newTeam\":\"HJK\",
\"playerName\":\"Zidane\",
\"transferStatus\":\"**Verified**\"}"}

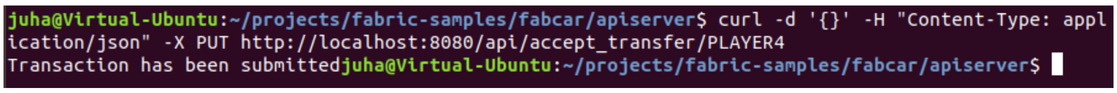


```
juha@Virtual-Ubuntu:~/projects/fabric-samples/fabcar/apiserver$ curl http://localhost:8080/api/quer
yplayer/PLAYER4
{"response":"{\"currentAssociation\":\"French Football Federation\",\"currentTeam\":\"Cannes\",\"do
cType\":\"player\",\"newAssociation\":\"Football Association of Finland\",\"newTeam\":\"HJK\",\"pla
yerName\":\"Zidane\",\"transferStatus\":\"Verified\"}"}juha@Virtual-Ubuntu:~/projects/fabric-sample
s/fabcar/apiserver$
```

7. Function for changing the transfer status to “Accepted” (id: PLAYER4)

juha@Virtual-Ubuntu:~/projects/fabric-samples/fabcar/apiserver$ **curl -d '{}' -H "Content-Type: application/json" -X PUT http://localhost:8080/api/accept_transfer/PLAYER4**

Transaction has been submitted

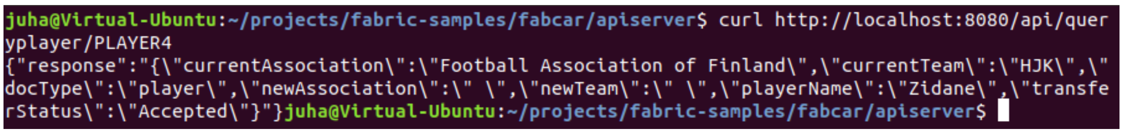


```
juha@Virtual-Ubuntu:~/projects/fabric-samples/fabcar/apiserver$ curl -d '{}' -H "Content-Type: appl
ication/json" -X PUT http://localhost:8080/api/accept_transfer/PLAYER4
Transaction has been submittedjuha@Virtual-Ubuntu:~/projects/fabric-samples/fabcar/apiserver$
```

8. Function for returning the information for one given player (id: PLAYER4)

juha@Virtual-Ubuntu:~/projects/fabric-samples/fabcar/apiserver$ **curl http://localhost:8080/api/queryplayer/PLAYER4**

{"response":"{
\"currentAssociation\":\"**Football Association of Finland**\",
\"currentTeam\":\"**HJK**\",
\"docType\":\"player\",
\"newAssociation\":\" \",
\"newTeam\":\" \",
\"playerName\":\"Zidane\",
\"transferStatus\":\"**Accepted**\"}"}

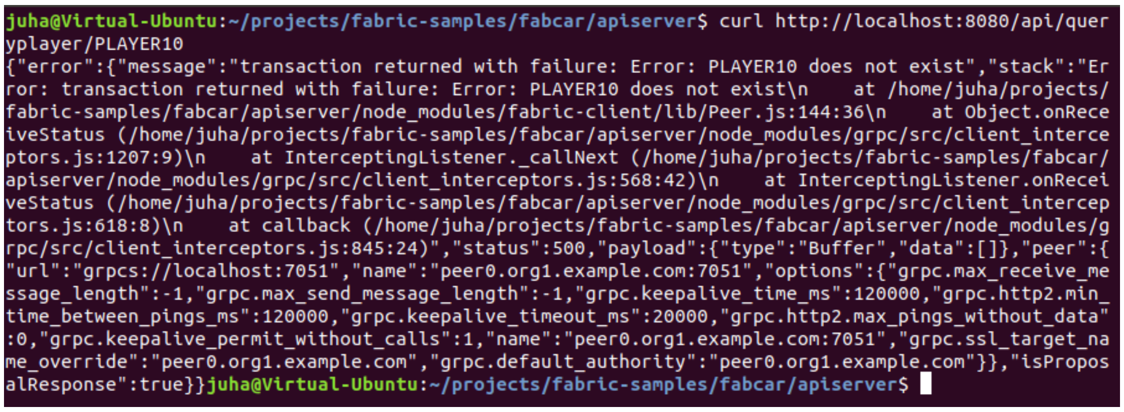


```
juha@Virtual-Ubuntu:~/projects/fabric-samples/fabcar/apiserver$ curl http://localhost:8080/api/quer
yplayer/PLAYER4
{"response":"{\"currentAssociation\":\"Football Association of Finland\",\"currentTeam\":\"HJK\",\"
docType\":\"player\",\"newAssociation\":\" \",\"newTeam\":\" \",\"playerName\":\"Zidane\",\"transfe
rStatus\":\"Accepted\"}"}juha@Virtual-Ubuntu:~/projects/fabric-samples/fabcar/apiserver$
```

9. Function for returning the information for one given player (id: PLAYER10)

juha@Virtual-Ubuntu:~/projects/fabric-samples/fabcar/apiserver$ **curl http://localhost:8080/api/queryplayer/PLAYER10**

{"error":{"message":"transaction returned with failure: Error: **PLAYER10 does not exist**……..}}

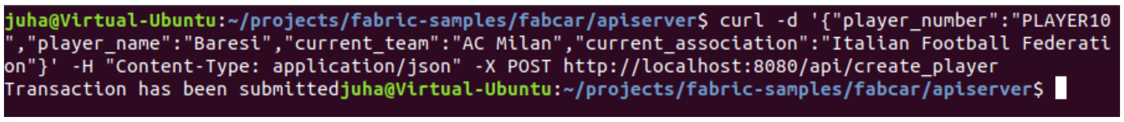


```
juha@Virtual-Ubuntu:~/projects/fabric-samples/fabcar/apiserver$ curl http://localhost:8080/api/quer
yplayer/PLAYER10
{"error":{"message":"transaction returned with failure: Error: PLAYER10 does not exist","stack":"Er
ror: transaction returned with failure: Error: PLAYER10 does not exist\n    at /home/juha/projects/
fabric-samples/fabcar/apiserver/node_modules/fabric-client/lib/Peer.js:144:36\n    at Object.onRece
iveStatus (/home/juha/projects/fabric-samples/fabcar/apiserver/node_modules/grpc/src/client_interce
ptors.js:1207:9)\n    at InterceptingListener._callNext (/home/juha/projects/fabric-samples/fabcar/
apiserver/node_modules/grpc/src/client_interceptors.js:568:42)\n    at InterceptingListener.onRecei
veStatus (/home/juha/projects/fabric-samples/fabcar/apiserver/node_modules/grpc/src/client_intercep
tors.js:618:8)\n    at callback (/home/juha/projects/fabric-samples/fabcar/apiserver/node_modules/g
rpc/src/client_interceptors.js:845:24)","status":500,"payload":{"type":"Buffer","data":[]},"peer":{
"url":"grpcs://localhost:7051","name":"peer0.org1.example.com:7051","options":{"grpc.max_receive_me
ssage_length":-1,"grpc.max_send_message_length":-1,"grpc.keepalive_time_ms":120000,"grpc.http2.min_
time_between_pings_ms":120000,"grpc.keepalive_timeout_ms":20000,"grpc.http2.max_pings_without_data"
:0,"grpc.keepalive_permit_without_calls":1,"name":"peer0.org1.example.com:7051","grpc.ssl_target_na
me_override":"peer0.org1.example.com","grpc.default_authority":"peer0.org1.example.com"}},"isPropos
alResponse":true}}juha@Virtual-Ubuntu:~/projects/fabric-samples/fabcar/apiserver$
```

10. Function for adding a new player (id: PLAYER10)

juha@Virtual-Ubuntu:~/projects/fabric-samples/fabcar/apiserver$ **curl -d '{"player_number":"PLAYER10","player_name":"Baresi","current_team":"AC Milan","current_association":"Italian Football Federation"}' -H "Content-Type: application/json" -X POST http://localhost:8080/api/create_player**

Transaction has been submitted

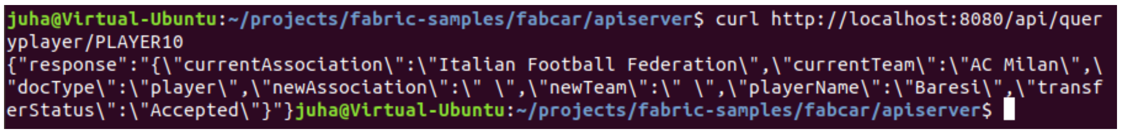


```
juha@Virtual-Ubuntu:~/projects/fabric-samples/fabcar/apiserver$ curl -d '{"player_number":"PLAYER10
","player_name":"Baresi","current_team":"AC Milan","current_association":"Italian Football Federati
on"}' -H "Content-Type: application/json" -X POST http://localhost:8080/api/create_player
Transaction has been submittedjuha@Virtual-Ubuntu:~/projects/fabric-samples/fabcar/apiserver$
```

11. Function for returning the information for one given player (id: PLAYER10)

juha@Virtual-Ubuntu:~/projects/fabric-samples/fabcar/apiserver$ **curl http://localhost:8080/api/queryplayer/PLAYER10**

```
{"response":"{
                \"currentAssociation\":\"Italian Football Federation\",
                \"currentTeam\":\"AC Milan\",
                \"docType\":\"player\",
                \"newAssociation\":\" \",
                \"newTeam\":\" \",
                \"playerName\":\"Baresi\",
                \"transferStatus\":\"Accepted\"}"}
```



```
juha@Virtual-Ubuntu:~/projects/fabric-samples/fabcar/apiserver$ curl http://localhost:8080/api/quer
yplayer/PLAYER10
{"response":"{\"currentAssociation\":\"Italian Football Federation\",\"currentTeam\":\"AC Milan\",\
"docType\":\"player\",\"newAssociation\":\" \",\"newTeam\":\" \",\"playerName\":\"Baresi\",\"transf
erStatus\":\"Accepted\"}"}juha@Virtual-Ubuntu:~/projects/fabric-samples/fabcar/apiserver$
```