

A Project Review - 1 Report
On
Payment Processing System using Blockchain

Submitted in partial fulfillment of the
Requirement for the award of the degree of

B.Tech with Specialization in AI/ML



Under the supervision of:
Ravindra Kumar Chahar

Submitted By:
Sakshi Jain (18SCSE180014)
Rakshit Singh (18SCSE1180060)

**SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GALGOTIAS UNIVERSITY, GREATER NOIDA
INDIA**

Declaration

I, the undersigned solemnly declare that the project report is based on our own work carried out during the course of our study under the supervision of Mr. Ravindra Kumar Chahar. I assert the statements made and conclusions drawn are an outcome of my research work. I further certify that

The work contained in the report is original and has been done by me under the general supervision of my supervisor.

The work has not been submitted to any other Institution for any other degree/diploma/certificate in this university or any other University of India or abroad.

We have followed the guidelines provided by the university in writing the report. Whenever we have used materials (data, theoretical analysis, and text) from other sources, we have given due credit to them in the text of the report and giving their details in the references.

Sakshi Jain (18SCSE180014)

Rakshit Singh (18SCSE1180060)

Bonafide Certificate

Certified that this project report “Payment Processing System using Blockchain” is the Bonafide work of Sakshi Jain and Rakshit Singh who carried out the project work under my supervision.

SUPERVISOR:

Ravindra Kumar Chahar

(Professor)

Statement of Project Report Preparation

- Thesis title: Payment Processing System using Blockchain
- Degree for which the report is submitted: B. Tech in CSE
- Project Supervisor was referred to for preparing the report.
- Specifications regarding thesis format have been closely followed.
- The contents of the thesis have been organized based on the guidelines.
- The report has been prepared without resorting to plagiarism.
- All sources used have been cited appropriately.
- The report has not been submitted elsewhere for a degree.

Acknowledgment

The success and result outcome of this project “Payment Processing System using Blockchain” required a lot of guidance and assistance from many people and I am extremely privileged to have got this all along the completion of my project. All that I have done is only due to such supervision and assistance and I would not forget to thank them.

I respect and thank our guide: Mr. Ravindra Kumar Chahar, for providing me an opportunity to do the project work under their guidance and giving us all support and guidance which made me complete the project duly. I am extremely thankful to my college Galgotias University for providing such a nice support and guidance through our teachers.

Index

Contents

Declaration	2
Bonafide Certificate	3
Statement of Project Report Preparation	4
Acknowledgment	5
Abstract	7
Introduction to Blockchain.....	8
Overview of E-Payment Methods and their Disadvantages	13
Hyperledger Fabric	15
Hyperledger Fabrics vs Traditional Blockchains	17
Usage in Payment Processing.....	25
Requirements.....	31
React JS.....	32
The Project UI	37
Deploying a Production Network.....	38
Conclusion	49
References.....	50

Abstract

The world of e-commerce and m-commerce is an ever-expanding one in today's age and with the number of online transactions reaching a record high. So is the need to secure the way people make these transactions. The traditional method uses some negotiable instruments such as drafts, debit cards, credit cards, electronic funds, direct credits, direct debits, internet banking, etc. A normal payment processing system is normally more prone to exploits like card theft and other types of fraud.

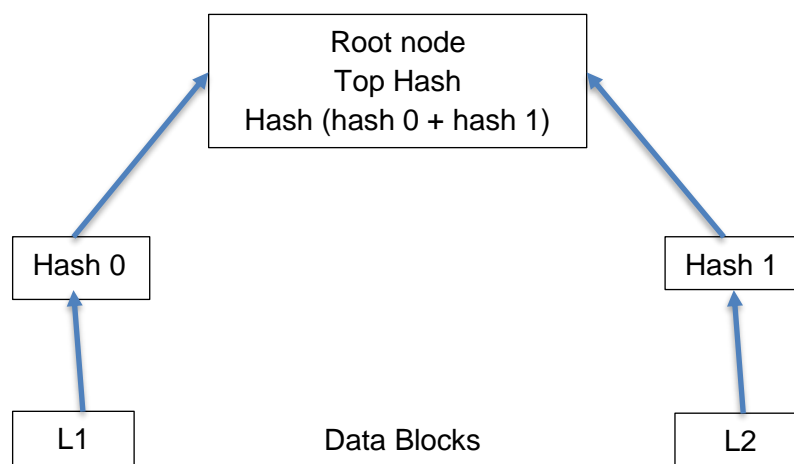
This calls for the need for a more secure and fraud-proof payment system. This is where the need of blockchain comes in. The power of cryptocurrency and blockchain in the payment system can do wonders. Payments made on a blockchain are fast, secure and cheap and facilitate international payment processing services through the use of encrypted distributed ledgers that provide trusted real-time verification of transactions without the need for intermediaries such as correspondent banks.

The result of this introduction in payment systems improves it by making it more secure and private with no middle man in between to break the end-to-end encryption.

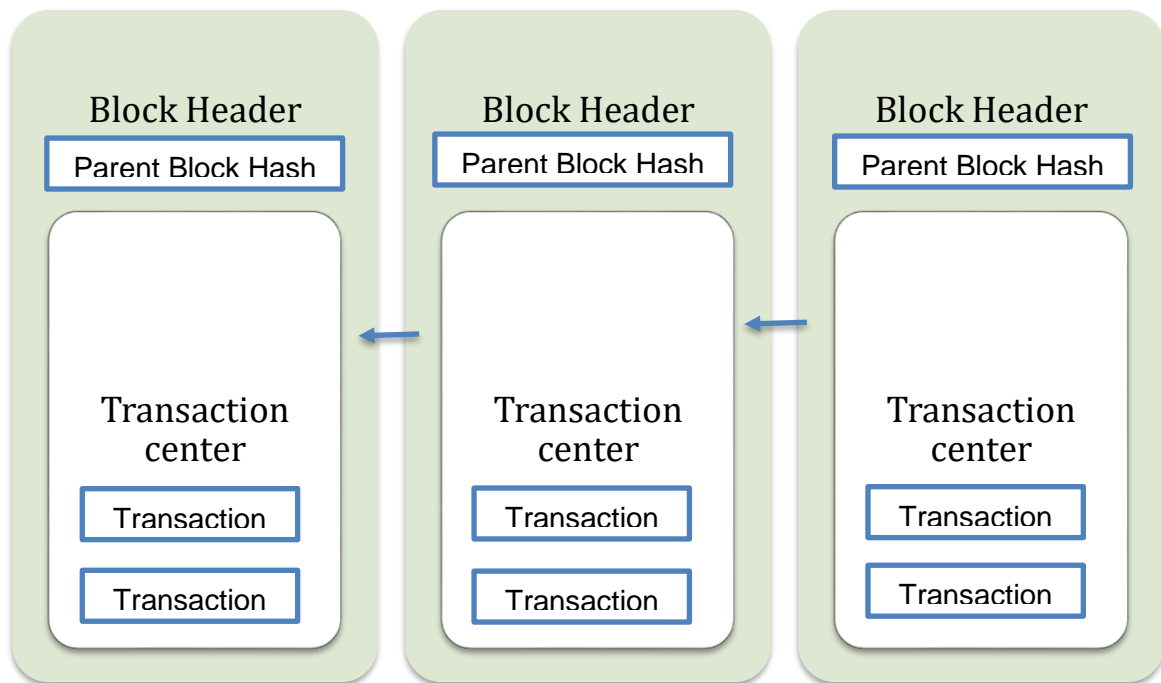
Our aim in this project is to come up with a system for payment processing that removes the demerits of a conventional payment processing system and brings in the merits of blockchain (DLT). The payment system developed using blockchain can be used in other areas for payments.

Introduction to Blockchain

A blockchain is a distributed peer to peer linked structure. It can be used to solve many problems such as maintaining ledger and order of all transactions. It is evolved from Merkle Tree. It is a tree in which every node has cryptographic hash of its children. It allows efficient and secure verification of contents in a large data structure.



Blockchain is a fully decentralized register that keeps track of all data exchanges in a secure manner. Each transaction is placed in a cryptographic block. Chain develops as new blocks are mounted to the chain as it develops into data storage structure. Sequence of blocks that holds entire list of dealing as public ledger. A block has a block header and Parent hash will point towards its parent. It stores transactions. There are other values also that are stored in the block header with parent block has such as block version, time stamp, nonce, merkle tree root has etc. The primary block which has no parent is called genesis block.



Blockchain is an emerging technology. In recent years, it is targeting almost every field revolving around humans.

Applications of blockchain

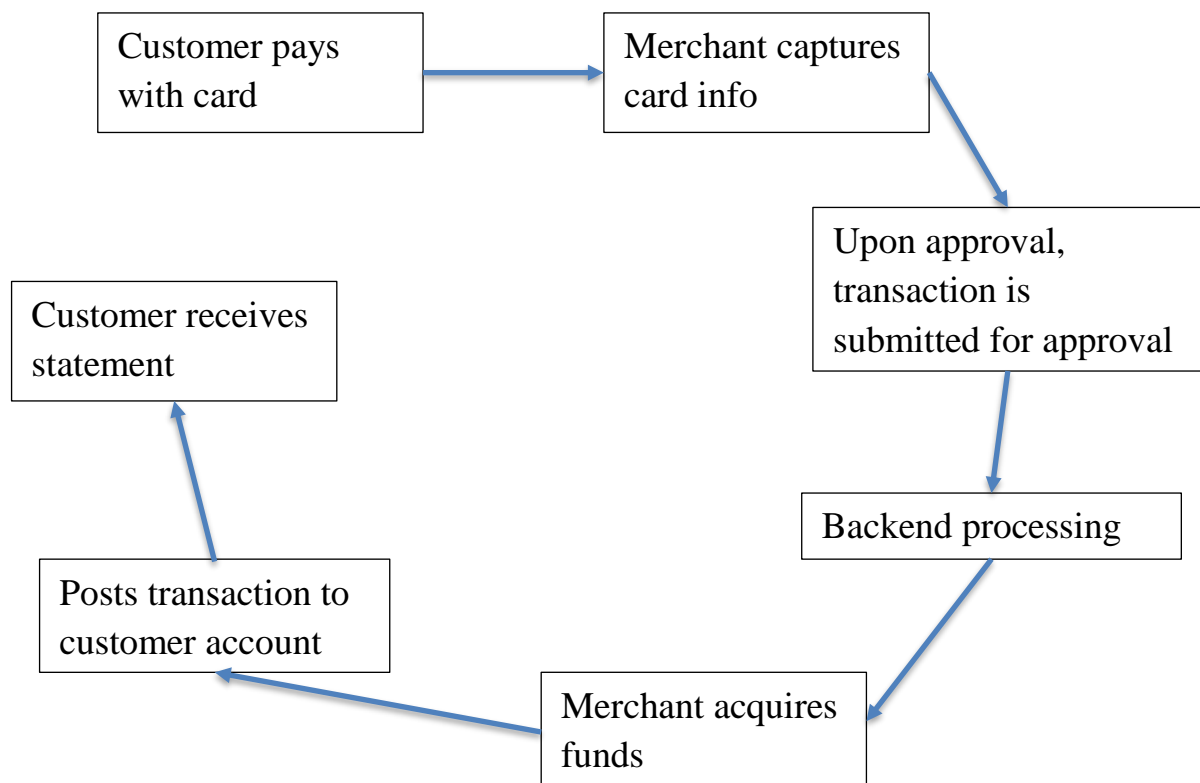
- Money transfer
- Financial exchanges
- Lending
- Insurance
- Real estate
- Secure personal information
- Voting
- Government benefits
- Securely sharing medical information
- Artist royalties

As the people are undergoing digital movement, the payment sector has adopted new and secured technology to ease out the payment processes. Until recently, people only accepted cash in exchange of good or services, but we have started building cashless economies. With advancement in technology we started moving towards payments via blockchain. It helps us to facilitate and process and most importantly verify payments on blockchain.

Smart contract have helped enormously by:

- Reducing time taken for payment
- Instant payments
- Automation of payments

Most companies have shifted their traditional payment system with new and improved technology. Companies like Veem, Circle and Robinhood uses blockchain for payment system.

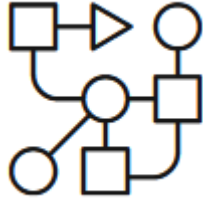


Blockchain security is a comprehensive risk management system for a blockchain network, using cybersecurity frameworks, assurance services and best practices to reduce risks against attacks and fraud.

Blockchain technology produces a structure of data with inherent security qualities. It's based on principles of cryptography, decentralization and consensus, which ensure trust in transactions. In most blockchains or distributed ledger technologies (DLT), the data is structured into blocks and each block contains a transaction or bundle of transactions. Each new block connects to all the blocks before it in a cryptographic chain in such a way that it's nearly impossible to tamper with. All transactions within the blocks are validated and agreed upon by a consensus mechanism, ensuring that each transaction is true and correct.

Blockchain technology enables decentralization through the participation of members across a distributed network. There is no single point of failure and a single user cannot change the record of transactions. However, blockchain technologies differ in some critical security aspects.

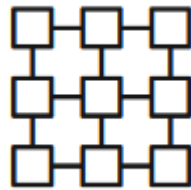
When building a blockchain application, it's critical to assess which type of network will best suit your business goals. Private and permissioned networks can be tightly controlled and preferable for compliance and regulatory reasons. However, public and permissionless networks can achieve greater decentralization and distribution.



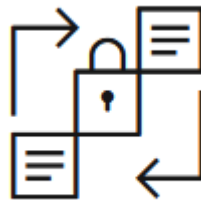
Public blockchains are public, and anyone can join them and validate transactions.



Private blockchains are restricted and usually limited to business networks. A single entity, or consortium, controls membership.



Permissionless blockchains have no restrictions on processors.



Permissioned blockchains are limited to a select set of users who are granted identities using certificates.

One proposed system provides a secure way to transfer payment and offers CRUD (Create, Read, Update and Deletion) operations

A V2G networks are proposed as both energy supply and consumers. It enables data sharing while securing sensitive user information. It ensures the anonymity of user payment data while enabling payment auditing by privileged users.

Overview of E-Payment Methods and their Disadvantages

1. Credit Card

Credit cards offer you a line of credit that can be used to make purchases, balance transfers and/or cash advances and requiring that you pay back the loan amount in the future. ... This money is not a loan, and no interest is charged. You will not have to make any minimum monthly payments.

Disadvantages:

- High rates of interest. If you fail to pay outstanding dues on credit cards within the due date, you will incur high interest rates. You can avoid paying additional interest by making timely repayments every month.
- Overspending. The ease of using credit cards often leads to overspending.

2. Bank Transfer

Bank transfer (or wire transfer) is a payment method that allows consumers to transfer money to a bank account around the world. The consumer is provided with a unique reference number and details of the bank account where they can make their payment.

Disadvantages:

- The order is not always a completed payment.
- There are potential delays in the payment completion process.
- There is the potential for payments that are not completed by the customer because the customer must contact the bank to complete the payment.

3. Digital Wallet

A digital wallet (or e-wallet) is a software-based system that securely stores users' payment information and passwords for numerous payment methods and websites. By using a digital wallet, users can complete purchases easily and quickly with near-field communications technology.

Disadvantages

- Limited retailers. The number of retailers that accept payments from an electronic wallet depends on the actual wallet you choose.
- Charges. There are some apps that might charge you for doing a transaction.
- Support Technology.

Payment Methods	Weaknesses
e-credit payments card	It is hard to implement a sustainable system of anonymity with data that are hard to trace. Data security is necessary in order to prevent fraud and allow disputes of transaction
e-cash payments	Overlooked due the popularity of e-credit. It can take longer for transactions to settle. The speed of settlement and disbursement also depends on the processor.
e-check payments	These transactions can't provide anonymity and have security issues due to fraudulent activities.

Hyperledger Fabric

Hyperledger Fabric is an open source enterprise-grade permissioned distributed ledger technology platform, designed for use in enterprise contexts that delivers some key differentiating capabilities over other popular distributed ledger or blockchain platforms.

The combination of different design features makes Fabric one of the better performing platforms available today both in terms of transaction processing and transaction conformation latency, and it enables privacy and confidentiality of transactions and the smart contracts (or “chaincode”) that implement them

Hyperledger Foundation hosts a number of enterprise-grade blockchain software projects. The projects are conceived and built by the developer community for vendors, end user organizations, service providers, start-ups, academics and others to use to build and deploy blockchain networks or commercial solutions. The Hyperledger Foundation staff is part of a larger Linux Foundation team that has years of experience in providing program management services for open source projects.

Companies currently using:

- USAA
- DTCC
- Amazon
- myGwok
- State Street Global Advisor

All Hyperledger projects follow the same design philosophy:



Modular



Highly secure



Interoperable



Cryptocurrency-Agnostic



Complete with APIs

Hyperledger Fabrics vs Traditional Blockchains

1. Modularity

Hyperledger Fabric has been specifically architected to have a modular architecture. Whether it's pluggable consensus, pluggable identity management protocols, key management protocols or cryptographic libraries, it has been designed at its core to be configured to meet enterprise use case requirements.

While Blockchain is a concept which can be implemented in many ways. It is basically a technology that stores data, on the other hand hyperledger uses blockchain as its database with another logic which is platform or framework dependent. Hyperledger transactions are stored in the blockchain nodes.

Following are the modular components that Fabric is comprised of:

- A pluggable ordering service establishes consensus on the order of transactions and then broadcasts blocks to peers.
- A pluggable membership service provider, which is responsible for associating entities in the network with cryptographic identities.
- An optional peer-to-peer gossip service that disseminates the blocks output by ordering service to other peer nodes.
- Smart contracts ("chaincode") run within a container environment for isolation and can be written in standard programming languages but do not have direct access to the ledger state.
- The ledger can be used to support a variety of DBMSs.
- A pluggable endorsement and validation policy enforcement that can be independently configured per application.

2. Permissioned vs Permissionless Blockchain

Permissionless blockchain, allow virtually anyone to participate and every participant is anonymous. In such a situation, there can be no trust other than that the state of the blockchain, prior to a certain depth, is immutable. In order to mitigate this absence of trust, permissionless blockchains typically employ a

“mined” native cryptocurrency or transaction fees to provide economic incentive to offset the extraordinary costs of participating in a form of byzantine fault tolerant consensus based on “Proof of Work”.

Whereas in a permissioned blockchain, its user belong to a set of known, identified and often vetted participants operating under a single governance model that yields a certain degree of trust. A permissioned blockchain provides a way to secure the interactions among a group of entities that have a common goal but which may not fully trust each other. By relying on the identities of the participants, a permissioned blockchain can use more traditional crash fault tolerant (CFT) or byzantine fault tolerant (BFT) consensus protocols that do not require costly mining.

3. Smart Contracts

A smart contract, or “chaincode”, act as a trusted distributed application that gains its security and trust from the blockchain and the underlying consensus among the peers it is the business logic of a blockchain application.

There are three key points that apply to smart contracts, especially when applied to a platform:

- Many smart contracts run concurrently in the network
- They may be deployed dynamically
- Application code should be treated as untrusted, potentially even malicious

Most existing smart-contracts capable blockchain platforms follow an order-execute architecture in which the consensus protocol:

- Validates and orders transactions then propagates them to all peer nodes
- Each peer then executes the transactions sequentially.

Smart contracts being used in a blockchain that works with the order-execute architecture must be deterministic, otherwise, consensus might never be reached. To address this issue, many platforms require that the smart contracts be written in non-standard, or domain-specific language so that non-deterministic operations can be eliminated. Further, since all transactions are executed one by one by all nodes, performance and scale is limited. The fact that the smart contract code

executes on every node in the system demands that complex measures be taken to protect the overall system from potentially malicious contracts in order to ensure resiliency of the overall system.

4. A New Approach

Fabric introduced a new architecture for transactions, called execute-order-validate. It addresses the resiliency, flexibility, faced by the order-execute model by separating the transaction flow into three steps:

- Execute a transaction and check if its correct, and thereby endorse it
- Order transactions via a consensus protocol
- Validate transactions against an application-specific endorsement policy before committing them to the ledger.

This design departs radically from the order-execute paradigm in that Fabric executes transactions before reaching final agreement on their order.

In Fabric, an application-specific endorsement policy specifies which peer nodes, or how many of them, need to vouch for the correct execution of a given smart contract. Thus, each transaction need only be executed by the subset of the peer nodes necessary to satisfy the transaction's endorsement policy. This allows for parallel execution increasing overall performance and scale of the system. This first phase also eliminates any non-determinism, as inconsistent results can be filtered out before ordering.

Because we have eliminated non-determinism, Fabric is the first blockchain technology that enables use of standard programming languages.

- Validates and orders transactions then propagates them to all peer nodes,
- Each peer then executes the transactions sequentially.

Smart contracts being used in a blockchain that works with the order-execute architecture must be deterministic, otherwise, consensus might never be reached. To address this issue, many platforms require that the smart contracts be written in non-standard, or domain-specific language so that non-deterministic operations can be eliminated. Further, since all transactions are executed one by one by all nodes, performance and scale is limited. The fact that the smart contract code executes on every node in the system demands that complex measures be taken to protect the overall system from potentially malicious contracts in order to ensure resiliency of the overall system.

Hyperledger Fabric Model

The key design features woven into Hyperledger Fabric that fulfill its promise of a comprehensive, yet customizable, enterprise blockchain solution:

- **Assets** — Asset definitions enable the exchange of almost anything with monetary value over the network, from whole foods to antique cars to currency futures.

Assets can range from the tangible (real estate and hardware) to the intangible (contracts and intellectual property). Hyperledger Fabric provides the ability to modify assets using chaincode transactions.

Assets are represented in Hyperledger Fabric as a collection of key-value pairs, with state changes recorded as transactions on a Channel ledger. Assets can be represented in binary and/or JSON form.

- **Chaincode** — Chaincode execution is partitioned from transaction ordering, limiting the required levels of trust and verification across node types, and optimizing network scalability and performance.

Chaincode is software defining an asset or assets, and the transaction instructions for modifying the asset(s); in other words, it's the business logic. Chaincode enforces the rules for reading or altering key-value pairs or other state database information. Chaincode functions execute against the ledger's current state database and are initiated through a transaction proposal. Chaincode execution results in a set of key-value writes (write set) that can be submitted to the network and applied to the ledger on all peers.

- **Ledger Features** — The immutable, shared ledger encodes the entire transaction history for each channel, and includes SQL-like query capability for efficient auditing and dispute resolution.

The ledger is the sequenced, tamper-resistant record of all state transitions in the fabric. State transitions are a result of chaincode invocations ('transactions') submitted by participating parties. Each transaction results in a set of asset key-value pairs that are committed to the ledger as creates, updates, or deletes.

The ledger is comprised of a blockchain ('chain') to store the immutable, sequenced record in blocks, as well as a state database to maintain current fabric

state. There is one ledger per channel. Each peer maintains a copy of the ledger for each channel of which they are a member.

Some features of a Fabric ledger:

- Query and update ledger using key-based lookups, range queries, and composite key queries
 - Read-only queries using a rich query language (if using CouchDB as state database)
 - Read-only history queries — Query ledger history for a key, enabling data provenance scenarios
 - Transactions consist of the versions of keys/values that were read in chaincode (read set) and keys/values that were written in chaincode (write set)
 - Transactions contain signatures of every endorsing peer and are submitted to ordering service
 - Transactions are ordered into blocks and are “delivered” from an ordering service to peers on a channel
 - Peers validate transactions against endorsement policies and enforce the policies
 - Prior to appending a block, a versioning check is performed to ensure that states for assets that were read have not changed since chaincode execution time
 - There is immutability once a transaction is validated and committed
 - A channel’s ledger contains a configuration block defining policies, access control lists, and other pertinent information. Channels contain Membership Service Provider instances allowing for crypto materials to be derived from different certificate authorities
- **Privacy** — Channels and private data collections enable private and confidential multi-lateral transactions that are usually required by competing businesses and regulated industries that exchange assets on a common network.

Hyperledger Fabric employs an immutable ledger on a per-channel basis, as well as chaincode that can manipulate and modify the current state of assets (i.e. update key-value pairs). A ledger exists in the scope of a channel — it can be shared across the entire network (assuming every participant is operating on one

common channel) — or it can be privatized to include only a specific set of participants.

In the latter scenario, these participants would create a separate channel and thereby isolate/segregate their transactions and ledger. In order to solve scenarios that want to bridge the gap between total transparency and privacy, chaincode can be installed only on peers that need to access the asset states to perform reads and writes (in other words, if a chaincode is not installed on a peer, it will not be able to properly interface with the ledger).

When a subset of organizations on that channel need to keep their transaction data confidential, a private data collection (collection) is used to segregate this data in a private database, logically separate from the channel ledger, accessible only to the authorized subset of organizations.

Thus, channels keep transactions private from the broader network whereas collections keep data private between subsets of organizations on the channel.

To further obfuscate the data, values within chaincode can be encrypted (in part or in total) using common cryptographic algorithms such as AES before sending transactions to the ordering service and appending blocks to the ledger. Once encrypted data has been written to the ledger, it can be decrypted only by a user in possession of the corresponding key that was used to generate the cipher text.

- **Security & Membership Services** — Permissioned membership provides a trusted blockchain network, where participants know that all transactions can be detected and traced by authorized regulators and auditors.

Hyperledger Fabric underpins a transactional network where all participants have known identities. Public Key Infrastructure is used to generate cryptographic certificates which are tied to organizations, network components, and end users or client applications. As a result, data access control can be manipulated and governed on the broader network and on channel levels. This “permissioned” notion of Hyperledger Fabric, coupled with the existence and capabilities of channels, helps address scenarios where privacy and confidentiality are paramount concerns.

- **Consensus** — It is a unique approach to consensus enables the flexibility and scalability needed for the enterprise.

In distributed ledger technology, consensus has recently become synonymous with a specific algorithm, within a single function. However, consensus

encompasses more than simply agreeing upon the order of transactions, and this differentiation is highlighted in Hyperledger Fabric through its fundamental role in the entire transaction flow, from proposal and endorsement, to ordering, validation and commitment. In a nutshell, consensus is defined as the full-circle verification of the correctness of a set of transactions comprising a block.

Consensus is achieved ultimately when the order and results of a block's transactions have met the explicit policy criteria checks. These checks and balances take place during the lifecycle of a transaction, and include the usage of endorsement policies to dictate which specific members must endorse a certain transaction class, as well as system chaincodes to ensure that these policies are enforced and upheld. Prior to commitment, the peers will employ these system chaincodes to make sure that enough endorsements are present, and that they were derived from the appropriate entities. Moreover, a versioning check will take place during which the current state of the ledger is agreed or consented upon, before any blocks containing transactions are appended to the ledger. This final check provides protection against double spend operations and other threats that might compromise data integrity, and allows for functions to be executed against non-static variables.

In addition to the multitude of endorsement, validity and versioning checks that take place, there are also ongoing identity verifications happening in all directions of the transaction flow. Access control lists are implemented on hierarchical layers of the network (ordering service down to channels), and payloads are repeatedly signed, verified and authenticated as a transaction proposal passes through the different architectural components. To conclude, consensus is not merely limited to the agreed upon order of a batch of transactions; rather, it is an overarching characterization that is achieved as a byproduct of the ongoing verifications that take place during a transaction's journey from proposal to commitment.

In addition to the horde of endorsements, validity and versioning checks that occur, identity verifications also occur most of the time in all directions of the transaction flow. Access management lists are enforced on hierarchic layers of the network, and payloads are repeatedly signed, verified and checked if they contain the required authentication as a transaction proposal passes through different parts of the network. To conclude, consensus isn't merely limited to the

agreed upon order of a batch of transactions; rather it's an overarching characterization that's achieved as a byproduct of the continued verifications that happen throughout a transaction's journey from proposal to commitment.

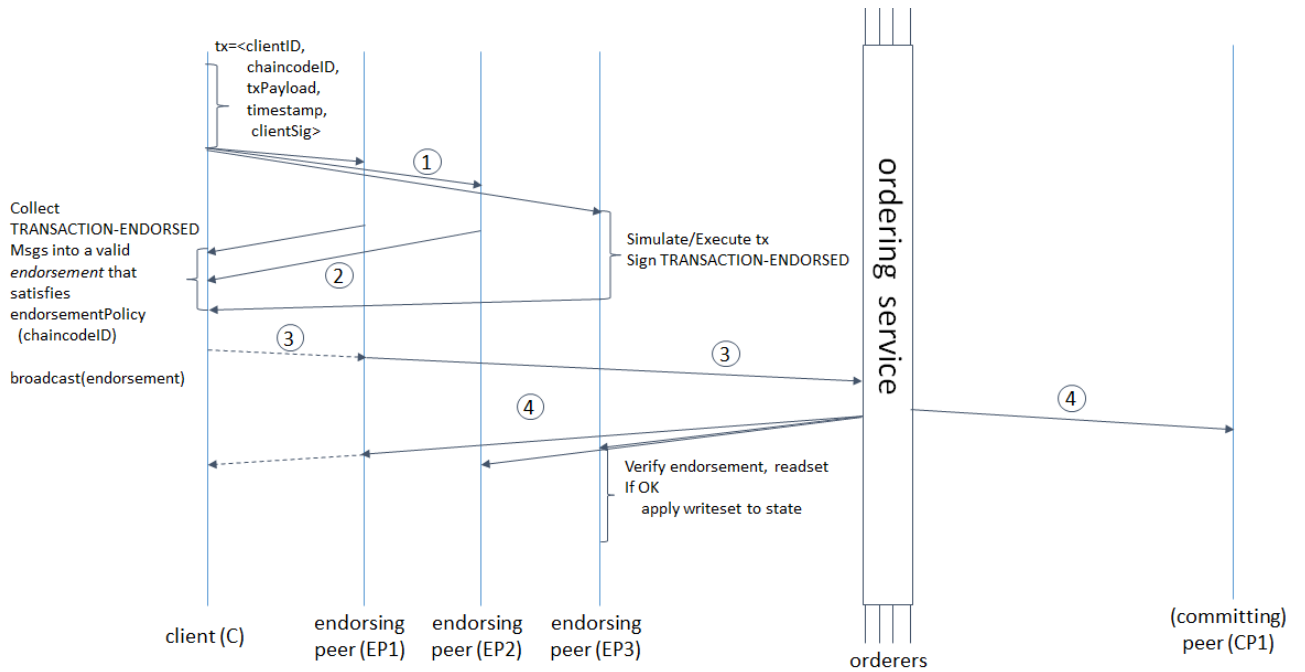


Fig: Swim lane sequence diagram of transaction flow.

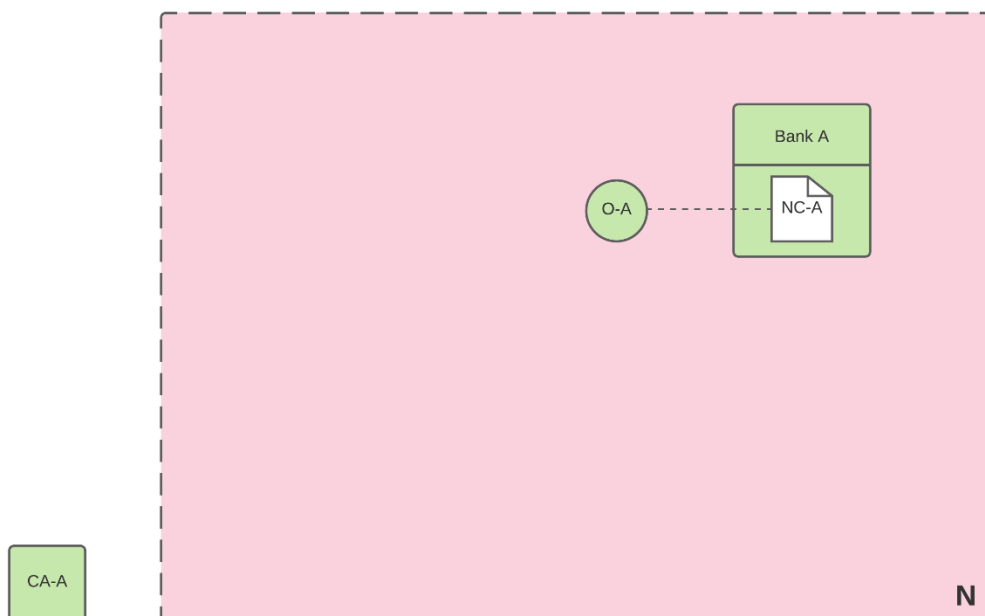
Usage in Payment Processing

So far we've read on why Hyperledger Fabric is a good fit in the domain of payment processing. To see how exactly we can use it to help create a Blockchain network that can make a difference in the field of payment processing we'll have to understand the working of a Hyperledger Fabric network with an example.

Let's suppose, four banks, A, B, C and D have jointly decided, and written into an agreement, that they will setup and use a Hyperledger Fabric network. Creating such a network will be divided into parts, as followed:

1. Setting up the network

A Hyperledger Fabric network is started when an order is created. In this example network, N the ordering service comprising a single node O-A is configured according to network configuration NC-A, which gives administrative rights to bank A. Certificate Authority CA-A is used to dispense identities to the administrators and network nodes of bank A.

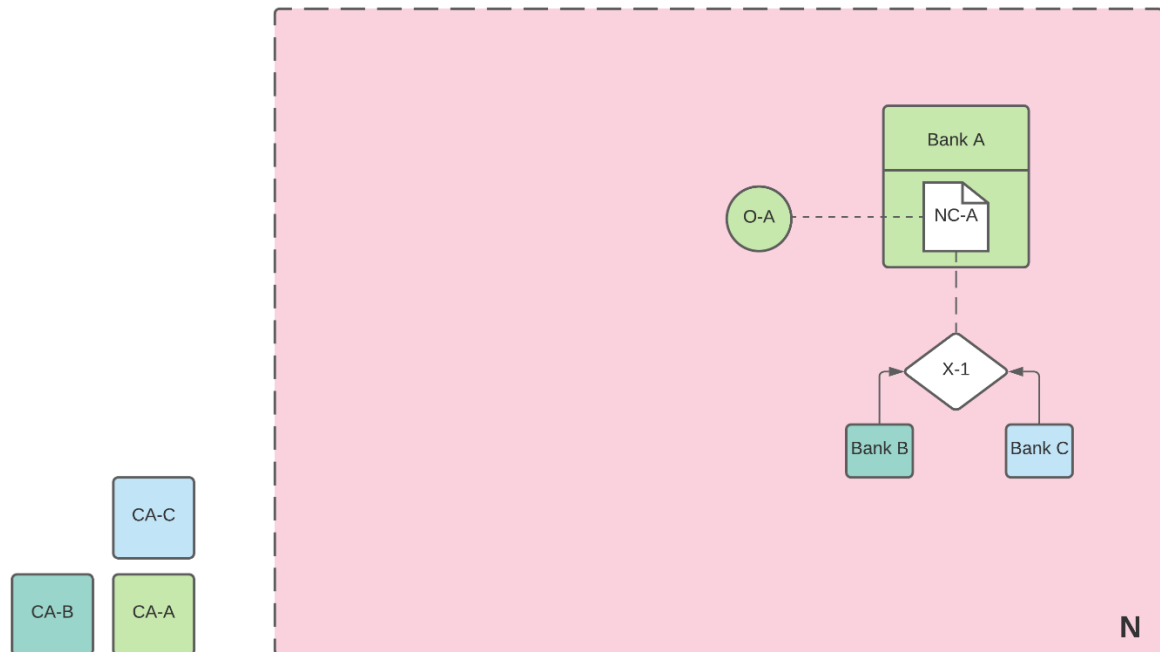


The first thing that defines a network is an ordering service, in this case O-A. The ordering service O-A is the initial administrative point for the network. As agreed beforehand, O-A is initially configured and started by an administrator in bank A, and hosted in A. The configuration NC-A contains the policies that describe

the starting set of administrative capabilities for the network. In the beginning, this is set to only give bank A rights over the network.

2. Defining a consortium

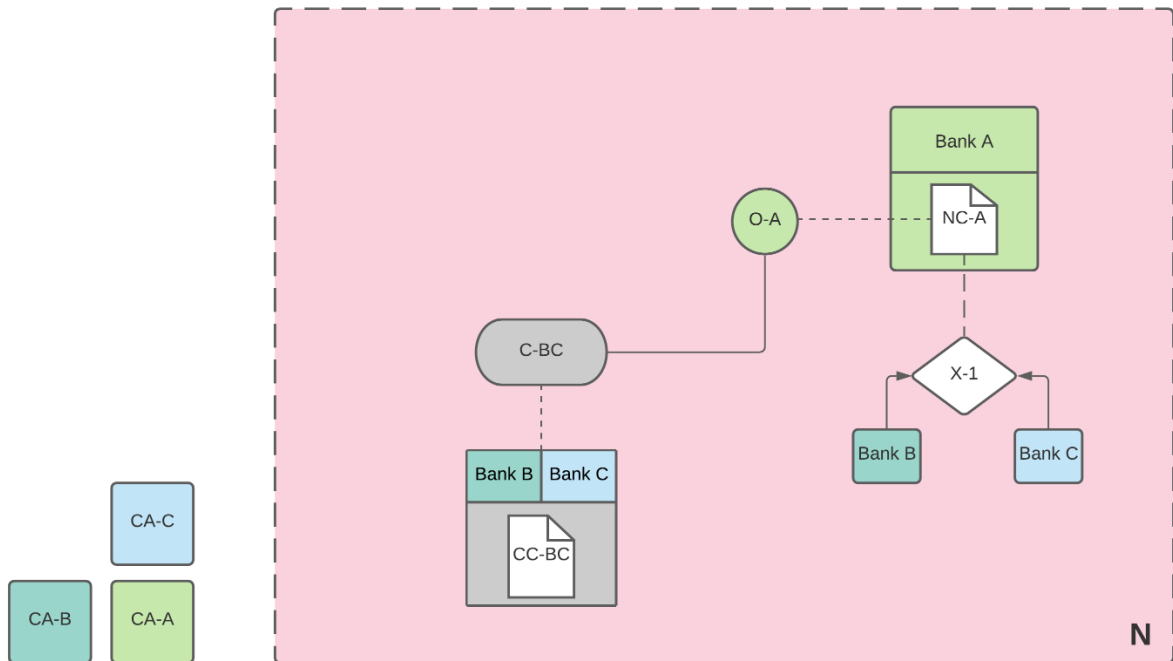
A consortium defines the set of organizations in the network who share a need to transact with one another.



Although the network can now be administered by bank A, there is not much that can be done on the network at this stage. The next step in creating the network will be to define a consortium. First banks B and C are added to the network as members along with their certificate authorities CA-B and CA-C, now, a network administrator (bank A in this case) can define a consortium X1 that contains the banks B and C. This consortium definition is stored in the network configuration NC-A, and will be used at the next stage of network development. The network, although started by a single bank/organization, is now controlled by a larger set of banks/organizations.

3. Creating a channel for consortium

A channel is the main communications mechanism by which the members of a consortium can communicate with each other.

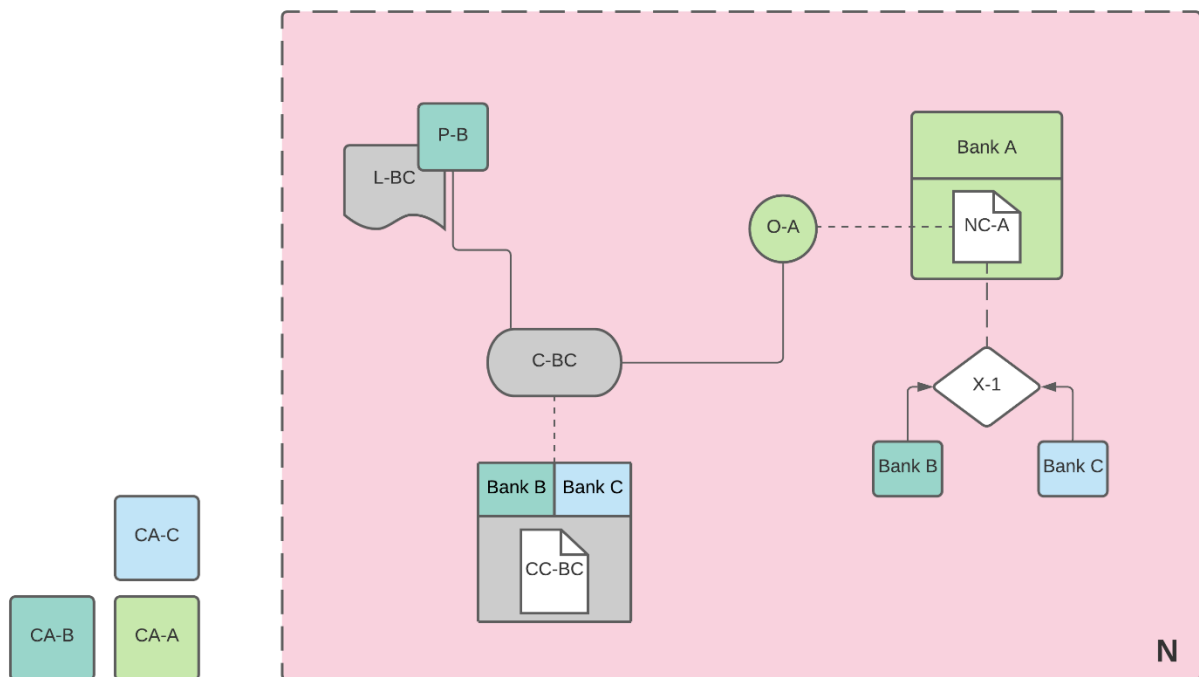


A channel C-BC will be created for B and C using the consortium definition X1. The channel is governed by a channel configuration CC-BC, completely separate from the network configuration. CC-BC is managed by B and C who have equal rights over C-BC and A has no rights in CC-BC whatsoever.

The channel C-BC provides a private communications mechanism for the consortium X1. We can see channel C-BC has been connected to the ordering service O-A but that nothing else is attached to it and even though channel C-BC is a part of the network N, it is quite separate from it. Also, bank A and D are not in this channel as it is for transaction processing between B and C only. Channels are therefore useful because they allow private communications between the participants constituting the channel. Also, the data in a channel is completely separated from the rest of the network and channels.

4. Peers and Ledgers

Our network N has acquired two new components, namely a peer node P-B and a ledger instance, L-A.

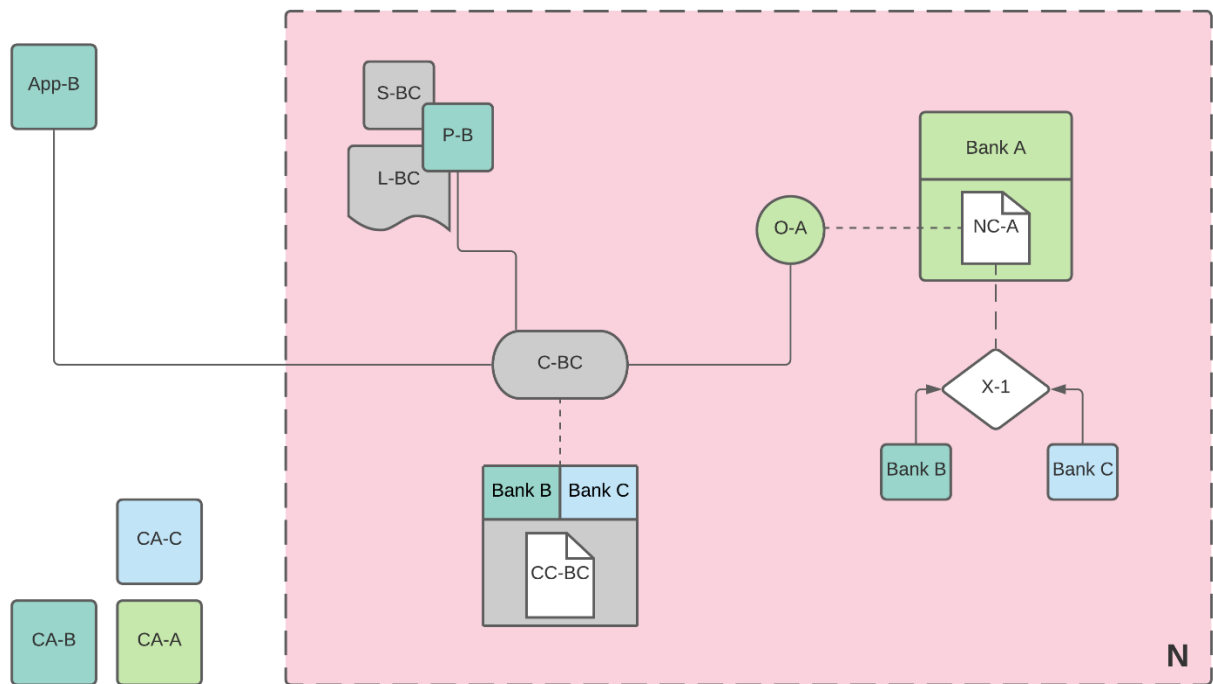


Peer nodes are the network components where copies of the blockchain ledger are hosted. P-B's purpose in the network is only to host a copy of the ledger L-B for others to access. We can think of L-B as being physically hosted on P-B, but logically hosted on the channel C-BC.

A key part of P-B's configuration is an X.509 identity issued by CA-B which associates P-B with bank B. When an administrator from bank A takes the action of joining peer P-B to channel C-BC, and the peer starts pulling blocks from the orderer O-A, the orderer uses the channel configuration CC-BC to determine P-B's permissions on this channel. For example, policy in CC-BC determines whether P-B (or bank A) can read and/or write on the channel C1.

5. Applications and Smart Contract chaincode

Now that channel C-BC has a ledger on it, we can start connecting client applications to allow users to transact on the network.



A smart contract S-B will be installed onto P-B. Client application App-B in bank B can use S-B to access the ledger via peer node P-B. App-B, P-B and O-A are all joined to channel C-BC, i.e. they can all make use of the communication facilities provided by the channel.

It might now appear that App-B can access the ledger L-B directly via P-B, but in fact, all access is managed by a special program called a smart contract chaincode, S-B. Think of S-B as defining all the common access patterns to the ledger. S-B provides a well-defined set of ways by which the ledger L-B can be queried or updated. In short, client application A-B has to go through smart contract S-B to get to ledger L-B.

Hyperledger Fabric mostly uses the term “chaincode”. Generally, smart contracts are used to define the transaction logic. Chaincode is just packaged smart contracts that can be deployed to a blockchain network.

6. Installing a chaincode package

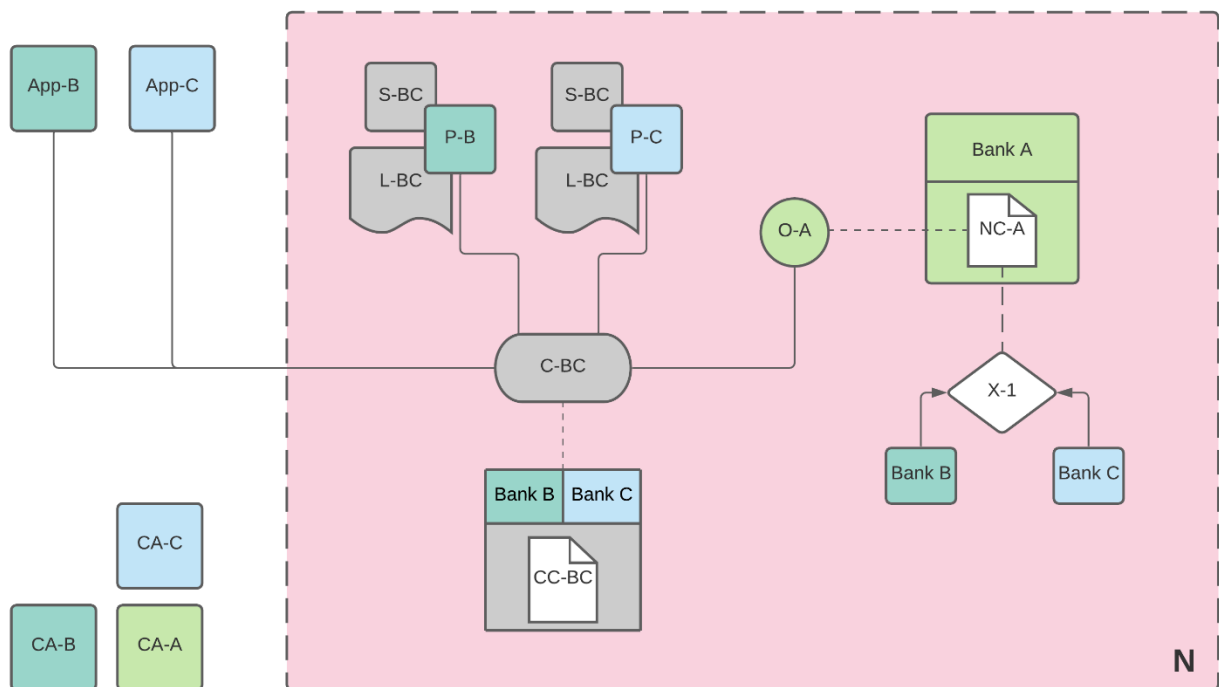
After a smart contract S-B has been developed, an administrator in bank B must create a chaincode package and deploy it onto peer node P-B. After this operation is completed, P-B has full knowledge of S-B. Specifically, P-B can see the

implementation logic of S-B, the program code that it uses to access the ledger L-B. If a bank has multiple peers on a single channel then it can select which peers to install the smart contracts on; it does not need to install a smart contract on every peer.

7. Defining a chaincode

Although a chaincode is installed on the peers of individual organizations, it is governed and operated in the scope of a channel. Each organization need to allow a chaincode definition, a set of params that establish how a chaincode will be used on a channel. An organization must allow a chaincode definition in order to use the installed smart contract to query the ledger and endorse transactions. In this example, which only has a single peer node P-B, an administrator in bank B must approve a chaincode definition for S-B.

An appropriate amount of organizations/banks need to approve a chaincode definition (by default, majority) before the chaincode definition can be committed to the channel and used to interact with the channel ledger. Because the channel only has one member, the administrator of A can commit the chaincode definition of S-B to the channel C-B. Once the definition has been committed, S-B can now be invoked by client application A-B.



Requirements

Operating System: Ubuntu

Common Requirements:

1. Git
2. cURL
3. Node.js
4. Npm

Backend Requirements:

1. Docker and Docker Compose
2. Go
3. SoftHSM
4. Express

Frontend Requirements:

1. React.js
2. Material-UI
3. Axios

React JS

React.js is an open-source JavaScript library that is used for building user interfaces specifically for single-page applications. It's used for handling the view layer for web and mobile apps. React also allows us to create reusable UI components. React was first created by Jordan Walke, a software engineer working for Facebook. React first deployed on Facebook's newsfeed in 2011 and on Instagram.com in 2012.

React allows developers to create large web applications that can change data, without reloading the page. The main purpose of React is to be fast, scalable, and simple. It works only on user interfaces in the application. This corresponds to the view in the MVC template. It can be used with a combination of other JavaScript libraries or frameworks, such as Angular JS in MVC.

React JS is also called simply to React or React.js.

React.js properties includes the following

- React.js is declarative
- React.js is simple
- React.js is component based
- React.js supports server side
- React.js is extensive
- React.js is fast
- React.js is easy to learn

JSX

In React, instead of using regular JavaScript for templating, it uses JSX. JSX is a simple JavaScript that allows HTML quoting and uses these HTML tag syntax to render subcomponents. HTML syntax is processed into JavaScript calls of React Framework. We can also write in pure old JavaScript.








Single-Way data flow

In React, a set of immutable values are passed to the components renderer as properties in its HTML tags. The component cannot directly modify any properties but can pass a call back function with the help of which we can do modifications. This complete process is known as “properties flow down; actions flow up”.












Virtual Document Object Model

React creates an in-memory data structure cache which computes the changes made and then updates the browser. This allows a special feature that enables the programmer to code as if the whole page is rendered on each change whereas react library only renders components that actually change.

Structure of a React JS project

 node_modules	12/18/2021 1:53 PM	File folder	
 public	12/18/2021 11:03 ...	File folder	
 src	12/18/2021 11:06 ...	File folder	
 .gitignore	10/26/1985 1:45 PM	Git Ignore Source ...	1 KB
 package	12/18/2021 12:10 ...	JSON Source File	1 KB
 package-lock	12/18/2021 12:10 ...	JSON Source File	1,121 KB
 README	10/26/1985 1:45 PM	Markdown Source ...	4 KB

The most common “src” folder looks somewhat like this:

 assets	12/18/2021 11:09 ...	File folder	
 Screens	12/18/2021 11:09 ...	File folder	
 App	10/26/1985 1:45 PM	CSS Source File	1 KB
 App	12/18/2021 12:30 ...	JavaScript Source ...	2 KB
 App.test	10/26/1985 1:45 PM	JavaScript Source ...	1 KB
 index	10/26/1985 1:45 PM	CSS Source File	1 KB
 index	10/26/1985 1:45 PM	JavaScript Source ...	1 KB
 logo	10/26/1985 1:45 PM	Firefox HTML Doc...	3 KB
 reportWebVitals	10/26/1985 1:45 PM	JavaScript Source ...	1 KB
 setupTests	10/26/1985 1:45 PM	JavaScript Source ...	1 KB
 style	12/18/2021 1:38 PM	JavaScript Source ...	1 KB

Why React?

1. Simplicity

React.js is just simpler to grasp right away. The component-based approach, well-defined lifecycle, and use of just plain JavaScript make React very simple to learn, build a professional web (and mobile applications), and support it. React uses a special syntax called JSX which allows you to mix HTML with JavaScript. This is not a requirement; Developer can still write in plain JavaScript but JSX is much easier to use.

2. Easy to learn

Anyone with a basic previous knowledge in programming can easily understand React while Angular and Ember are referred to as ‘Domain-specific Language’, implying that it is difficult to learn them. To react, you just need basic knowledge of CSS and HTML.

3. Native Approach

React can be used to create mobile applications (React Native). And React is a diehard fan of reusability, meaning extensive code reusability is supported. So at the same time, we can make IOS, Android and Web applications.

4. Data Binding

React uses one-way data binding and an application architecture called Flux controls the flow of data to components through one control point – the dispatcher. It's easier to debug self-contained components of large ReactJS apps.

5. Performance

React does not offer any concept of a built-in container for dependency. You can use Browserify, Require JS, EcmaScript 6 modules which we can use via Babel, ReactJS-di to inject dependencies automatically.

6. Testability

ReactJS applications are super easy to test. React views can be treated as functions of the state, so we can manipulate with the state we pass to the ReactJS view and take a look at the output and triggered actions, events, functions, etc.

How to run React JS project

1. Open your terminal and then type “git clone” {the url to the GitHub repo}
This clones the repo.
2. cd into the new folder and type “npm install”. This installs the required dependencies.
3. To run the React project type “npm start”.
4. The project runs on a local host

```
Compiled with warnings.

src\Screens\History.jsx
  Line 3:8: 'Box' is defined but never used      no-unused-vars
  Line 4:8: 'Container' is defined but never used  no-unused-vars

Search for the keywords to learn more about each warning.
To ignore, add // eslint-disable-next-line to the line before.

Assets by status 6.87 KiB [cached] 1 asset
Assets by chunk 2.67 MiB (name: main)
  asset static/js/bundle.js 2.66 MiB [emitted] (name: main) 1 related asset
  asset main.cddf0b08d88bbfff68aa.hot-update.js 13.4 KiB [emitted] [immutable] [hmr] (name: main) 1 related asset
Assets by path *.json 611 bytes
  asset asset-manifest.json 583 bytes [emitted]
  asset main.cddf0b08d88bbfff68aa.hot-update.json 28 bytes [emitted] [immutable] [hmr]
Asset index.html 1.67 KiB [emitted]
Entrypoint main 2.67 MiB (2.44 MiB) = static/js/bundle.js 2.66 MiB main.cddf0b08d88bbfff68aa.hot-update.js 13.4 KiB
Auxiliary assets
Cached modules 2.46 MiB [cached] 557 modules
Runtime modules 31.4 KiB 16 modules
./src/Screens/History.jsx 8.73 KiB [built] [code generated]

WARNING in src\Screens\History.jsx
  Line 3:8: 'Box' is defined but never used      no-unused-vars
  Line 4:8: 'Container' is defined but never used  no-unused-vars

webpack 5.65.0 compiled with 1 warning in 787 ms
```

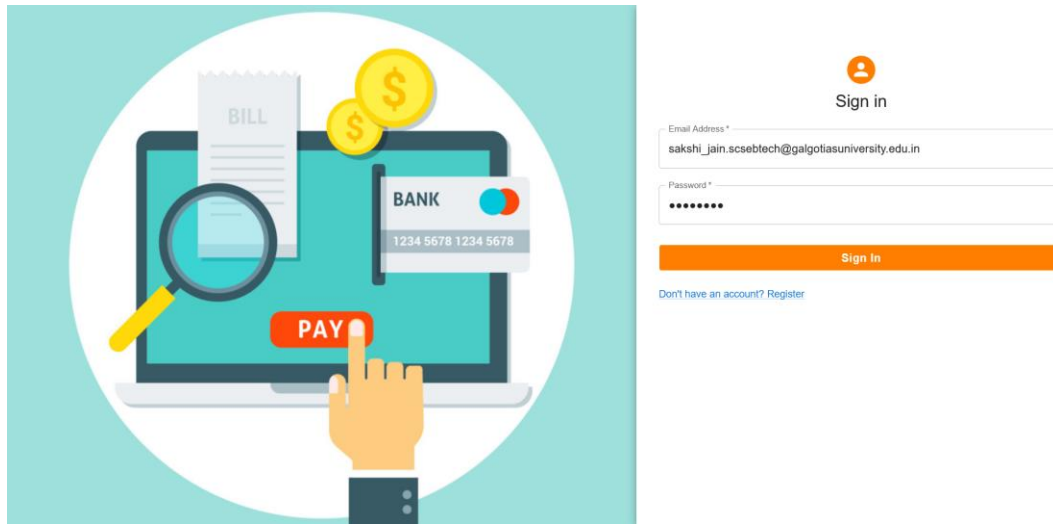
Fig : The output of “npm start” command.

The Project UI

The Login Page

User logs in using his credentials and all the user data is received from the API.

Output:



The Home Page:

User can pay to other and view transaction history

Output:

To *

Amount *

Pay

History

From	To	Date	Time	Amount
123	321	16 Mar, 2019	12:12 AM	Rs. 312.44
113	341	17 July, 2021	12:00 PM	Rs. 500
678	456	30 June, 2009	09:00 AM	Rs. 7.4
94	78954	20 Dec, 2019	5:00 AM	Rs. 12000
45467	099	15 Feb 2007	11:50 PM	Rs. 8000
123	321	16 Mar, 2019	12:12 AM	Rs. 312.44
113	341	17 July, 2021	12:00 PM	Rs. 500
678	456	30 June, 2009	09:00 AM	Rs. 7.4
94	78954	20 Dec, 2019	5:00 AM	Rs. 12000
45467	099	15 Feb 2007	11:50 PM	Rs. 8000

Deploying a Production Network

1. Decide on your network configuration

The structure of a blockchain network will be dictated by the use case it's serving. There are too many options to give definitive guidance on every point, but let's consider a few scenarios.

In contrast to development environments or proofs of concept, security, resource management, and high availability become a priority when operating in production. How many nodes do you need to satisfy high availability, and in what data centers do you wish to deploy them in to satisfy both the needs of disaster recovery and data residency? How will you ensure that your private keys and roots of trust remain secure?

In addition to the above, here is a sampling of the decisions you will need to make before deploying components:

- **Certificate Authority configuration.** As part of the overall decisions you have to make about your peers (how many, how many on each channel, and so on) and about your ordering service (how many nodes, who will own them), you also have to decide on how the CAs for your organization will be deployed. Production networks should be using Transport Layer Security (TLS), which will require setting up a TLS CA and using it to generate TLS certificates. This TLS CA will need to be deployed before your enrollment CA. We'll discuss this more in Step three: Set up your CAs.
- **Use Organizational Units or not?** Some organizations might find it necessary to establish Organizational Units to create a separation between certain identities and MSPs created by a single CA (for example, a manufacturer might want one organizational unit for its shipping department and another for its quality control department). Note that this is separate from the concept of the "Node OU", in which identities can have roles coded into them (for example, "admin" or "peer").
- **Database type.** Some channels in a network might require all data to be modeled in a way CouchDB as the State Database can understand, while

other networks, prioritizing speed, might decide that all peers will use LevelDB. Note that channels should not have peers that use both CouchDB and LevelDB on them, as CouchDB imposes some data restrictions on keys and values. Keys and values that are valid in LevelDB may not be valid in CouchDB.

- Channels and private data. Some networks might decide that Channels are the best way to ensure privacy and isolation for certain transactions. Others might decide that fewer channels, supplemented where necessary with Private data collections, better serves their privacy needs.
- Container orchestration. Different users might also make different decisions about their container orchestration, creating separate containers for their peer process, logging, CouchDB, gRPC communications, and chaincode, while other users might decide to combine some of these processes.
- Chaincode deployment method. Users have the option to deploy their chaincode using either the built in build and run support, a customized build and run using the External Builders and Launchers, or using an Chaincode as an external service.
- Using firewalls. In a production deployment, components belonging to one organization might need access to components from other organizations, necessitating the use of firewalls and advanced networking configuration. For example, applications using the Fabric SDK require access to all endorsing peers from all organizations and the ordering services for all channels. Similarly, peers need access to the ordering service on the channels that they are receiving new blocks from.

However and wherever your components are deployed, you will need a high degree of expertise in your management system of choice (such as Kubernetes) in order to efficiently operate your network. Similarly, the structure of the network must be designed to fit the business use case and any relevant laws and regulations government of the industry in which the network will be designed to function.

This deployment guide will not go through every iteration and potential network configuration, but does give common guidelines and rules to consider.

2. Set up a cluster for your resources

Generally speaking, Fabric is agnostic to the methods used to deploy and manage it. It is possible, for example, to deploy and manage a peer on a laptop. For a number of reasons, this is likely to be unadvisable, but there is nothing in Fabric that prohibits it.

As long as you have the ability to deploy containers, whether locally (or behind a firewall), or in a cloud, it should be possible to stand up components and connect them to each other. However, Kubernetes features a number of helpful tools that have made it a popular container management platform for deploying and managing Fabric networks. For more information about Kubernetes, check out the Kubernetes documentation. This topic will mostly limit its scope to the binaries and provide instructions that can be applied when using a Docker deployment or Kubernetes.

However and wherever you choose to deploy your components, you will need to make sure you have enough resources for the components to run effectively. The sizes you need will largely depend on your use case. If you plan to join a single peer to several high volume channels, it will need much more CPU and memory than if you only plan to join to a single channel. As a rough estimate, plan to dedicate approximately three times the resources to a peer as you plan to allocate to a single ordering node (as you will see below, it is recommended to deploy at least three and optimally five nodes in an ordering service). Similarly, you should need approximately a tenth of the resources for a CA as you will for a peer. You will also need to add storage to your cluster (some cloud providers may provide storage) as you cannot configure Persistent Volumes and Persistent Volume Claims without storage being set up with your cloud provider first. The use of persistent storage ensures that data such as MSPs, ledgers, and installed chaincodes are not stored on the container filesystem, preventing them from being destroyed if the containers are destroyed.

By deploying a proof of concept network and testing it under load, you will have a better sense of the resources you will require.

Managing your infrastructure

- The exact methods and tools you use to manage your backend will depend on the backend you choose. However, here are some considerations worth noting.

- Using secret objects to securely store important configuration files in your cluster. For information about Kubernetes secrets, check out Kubernetes secrets. You also have the option to use Hardware Security Modules (HSMs) or encrypted Persistent Volumes (PVs). Along similar lines, after deploying Fabric components, you will likely want to connect to a container on your own backend, for example using a private repo in a service like Docker Hub. In that case, you will need to code the login information in the form of a Kubernetes secret and include it in the YAML file when deploying components.
- Cluster considerations and node sizing. In step 2 above, we discussed a general outline for how to think about the sizings of nodes. Your use case, as well as a robust period of development, is the only way you will truly know how large your peers, ordering nodes, and CAs will need to be.
- How you choose to mount your volumes. It is a best practice to mount the volumes relevant to your nodes external to the place where your nodes are deployed. This will allow you to reference these volumes later on (for example, restarting a node or a container that has crashed) without having to redeploy or regenerate your crypto material.
- How you will monitor your resources. It is critical that you establish a strategy and method for monitoring the resources used by your individual nodes and the resources deployed to your cluster generally. As you join your peers to more channels, you will likely need to increase its CPU and memory allocation. Similarly, you will need to make sure you have enough storage space for your state database and blockchain.

3. Set up a cluster for your resources

The first component that must be deployed in a Fabric network is a CA. This is because the certificates associated with a node (not just for the node itself but also the certificates identifying who can administer the node) must be created before the node itself can be deployed. While it is not necessary to use the Fabric CA to create these certificates, the Fabric CA also creates MSP structures that are needed for components and organizations to be properly defined. If a user chooses to use a CA other than the Fabric CA, they will have to create the MSP folders themselves.

- One CA (or more, if you are using intermediate CAs — more on intermediate CAs below) is used to generate (through a process called “enrollment”) the certificates of the admin of an organization, the MSP of that organization, and any nodes owned by that organization. This CA will also generate the certificates for any additional users. Because of its role in “enrolling” identities, this CA is sometimes called the “enrollment CA” or the “ecert CA”.
- The other CA generates the certificates used to secure communications on Transport Layer Security (TLS). For this reason, this CA is often referred to as a “TLS CA”. These TLS certificates are attached to actions as a way of preventing “man in the middle” attacks. Note that the TLS CA is only used for issuing certificates for nodes and can be shut down when that activity is completed. Users have the option to use one way (client only) TLS as well as two way (server and client) TLS, with the latter also known as “mutual TLS”. Because specifying that your network will be using TLS (which is recommended) should be decided before deploying the “enrollment” CA (the YAML file specifying the configuration of this CA has a field for enabling TLS), you should deploy your TLS CA first and use its root certificate when bootstrapping your enrollment CA. This TLS certificate will also be used by the fabric-ca client when connecting to the enrollment CA to enroll identities for users and nodes.

While all of the non-TLS certificates associated with an organization can be created by a single “root” CA (that is, a CA that is its own root of trust), for added security organizations can decide to use “intermediate” CAs whose certificates are created by a root CA (or another intermediate CA that eventually leads back to a root CA). Because a compromise in the root CA leads to a collapse for its entire trust domain (the certs for the admins, nodes, and any CAs it has generated certificates for), intermediate CAs are a useful way to limit the exposure of the root CA. Whether you choose to use intermediate CAs will depend on the needs of your use case. They are not mandatory. Note that it is also possible to configure a Lightweight Directory Access Protocol (LDAP) to manage identities on a Fabric network for those enterprises that already have this implementation and do not want to add a layer of identity management to their existing infrastructure. The LDAP effectively pre registers all of the members of the directory and allows them to enroll based on the criteria given.

In a production network, it is recommended to deploy at least one CA per organization for enrollment purposes and another for TLS. For example, if

you deploy three peers that are associated with one organization and an ordering node that is associated with an ordering organization, you will need at least four CAs. Two of the CAs will be for the peer organization (generating the enrollment and TLS certificates for the peer, admins, communications, and the folder structure of the MSP representing the organization) and the other two will be for the orderer organization. Note that users will generally only register and enroll with the enrollment CA, while nodes will register and enroll with both the enrollment CA (where the node will get its signing certificates that identify it when it attempts to sign its actions) and with the TLS CA (where it will get the TLS certificates it uses to authenticate its communications).

For an example of how to setup an organization CA and a TLS CA and enroll their admin identity, check out the Fabric CA Deployment Guide. The deploy guide uses the Fabric CA client to register and enroll the identities that are required when setting up CAs.

4. Set up a cluster for your resources

After you have created your CAs, you can use them to create the certificates for the identities and components related to your organization (which is represented by an MSP). For each organization, you will need to, at a minimum:

- **Register and enroll an admin identity and create an MSP.** After the CA that will be associated with an organization has been created, it can be used to first register a user and then enroll an identity (producing the certificate pair used by all entities on the network). In the first step, a username and password for the identity is assigned by the admin of the CA. Attributes and affiliations can also be given to the identity (for example, a role of admin, which is necessary for organization admins). After the identity has been registered, it can be enrolled by using the username and password. The CA will generate two certificates for this identity — a public certificate (also known as a “signcert” or “public cert”) known to the other members of the network, and the private key (stored in the keystore folder) used to sign actions taken by the identity. The CA will also generate a set of folders called an “MSP” containing the public certificate of the CA issuing the certificate and the root of trust for the CA (this may or may not be the same CA). This MSP can be

thought of as defining the organization associated with the identity of the admin. In cases where the admin of the org will also be an admin of a node (which will be typical), you must create the org admin identity before creating the local MSP of a node, since the certificate of the node admin must be used when creating the local MSP.

- **Register and enroll node identities.** Just as an org admin identity is registered and enrolled, the identity of a node must be registered and enrolled with both an enrollment CA and a TLS CA (the latter generates certificates that are used to secure communications). Instead of giving a node a role of admin or user when registering it with the enrollment CA, give it a role of peer or orderer. As with the admin, attributes and affiliations for this identity can also be assigned. The MSP structure for a node is known as a “local MSP”, since the permissions assigned to the identities are only relevant at the local (node) level. This MSP is created when the node identity is created, and is used when bootstrapping the node.

For more conceptual information about identities and permissions in a Fabric-based blockchain network, see Identity and Membership Service Provider (MSP).

For more information about how to use a CA to register and enroll identities, including sample commands, check out Registering and enrolling identities with a CA.

5. Deploy peers and ordering nodes

Once you have gathered all of the certificates and MSPs you need, you’re almost ready to create a node. As discussed above, there are a number of valid ways to deploy nodes.

Before any node can be deployed, its configuration file must be customized. For the peer, this file is called `core.yaml`, while the configuration file for ordering nodes is called `orderer.yaml`.

You have three main options for tuning your configuration.

1. Edit the YAML file bundled with the binaries.
2. Use environment variable overrides when deploying.
3. Specify flags on CLI commands.

Option 1 has the advantage of persisting your changes whenever you bring down and bring back up the node. The downside is that you will have to port the options you customized to the new YAML when upgrading to a new binary version (you should use the latest YAML when upgrading to a new version).

Creating a peer

If you've read through the key concept topic on Peers, you should have a good idea of the role peers play in a network and the nature of their interactions with other network components. Peers are owned by organizations that are members of a channel (for this reason, these organizations are sometimes called "peer organizations"). They connect to the ordering service and to other peers, have smart contracts installed on them, and are where ledgers are stored.

These roles are important to understand before you create a peer, as they will influence your customization and deployment decisions. For a look at the various decisions you will need to make, check out [Planning for a production peer](#).

The configuration values in a peer's `core.yaml` file must be customized or overridden with environment variables. You can find the default `core.yaml` configuration file in the `sampleconfig` directory of Hyperledger Fabric. This configuration file is bundled with the peer image and is also included with the downloadable binaries. For information about how to download the production `core.yaml` along with the peer image, check out [Deploy the peer](#).

While there are many parameters in the default `core.yaml`, you will only need to customize a small percentage of them. In general, if you do not have the need to change a tuning value, keep the default value.

Among the parameters in `core.yaml`, there are:

- **Identifiers:** these include not just the paths to the relevant local MSP and Transport Layer Security (TLS) certificates, but also the name (known as the “peer ID”) of the peer and the MSP ID of the organization that owns the peer.
- **Addresses and paths:** because peers are not entities unto themselves but interact with other peers and components, you must specify a series of addresses in the configuration. These include addresses where the peer itself can be found by other components as well as the addresses where, for example, chaincodes can be found (if you are employing external chaincodes). Similarly, you will need to specify the location of your ledger (as well as your state database type) and the path to your external builders (again, if you intend to employ external chaincodes). These include Operations and metrics, which allow you to set up methods for monitoring the health and performance of your peer through the configuration of endpoints.
- **Gossip:** components in Fabric networks communicate with each other using the “gossip” protocol. Through this protocol, they can be discovered by the discovery service and disseminate blocks and private data to each other. Note that gossip communications are secured using TLS.

When you’re comfortable with how your peer has been configured, how your volumes are mounted, and your backend configuration, you can run the command to launch the peer (this command will depend on your backend configuration).

Deploying a production peer

- Planning for a production peer
- Checklist for a production peer
- Deploy the peer

Creating an ordering node

If you’ve read through the key concept topic on The Ordering Service, you should have a good idea of the role the ordering service plays in a network and the nature

of its interactions with other network components. The ordering service is responsible for literally “ordering” endorsed transactions into blocks, which peers then validate and commit to their ledgers.

These roles are important to understand before you create an ordering service, as it will influence your customization and deployment decisions. Among the chief differences between a peer and ordering service is that in a production network, multiple ordering nodes work together to form the “ordering service” of a channel. This creates a series of important decisions that need to be made at both the node level and at the cluster level. Some of these cluster decisions are not made in individual ordering node `orderer.yaml` files but instead in the `configtx.yaml` file that is used to generate the genesis block for the system channel (which is used to bootstrap ordering nodes), and also used to generate the genesis block of application channels. For a look at the various decisions you will need to make, check out [Planning for an ordering service](#).

The configuration values in an ordering node’s `orderer.yaml` file must be customized or overridden with environment variables. You can find the default `orderer.yaml` configuration file in the `sampleconfig` directory of Hyperledger Fabric.

This configuration file is bundled with the `orderer` image and is also included with the downloadable binaries. For information about how to download the production `orderer.yaml` along with the `orderer` image, check out [Deploy the ordering service](#).

While there are many parameters in the default `orderer.yaml`, you will only need to customize a small percentage of them. In general, if you do not have the need to change a tuning value, keep the default value.

Among the parameters in `orderer.yaml`, there are:

- **Identifiers:** these include not just the paths to the relevant local MSP and Transport Layer Security (TLS) certificates, but also the MSP ID of the organization that owns the ordering node.

- **Addresses and paths:** because ordering nodes interact with other components, you must specify a series of addresses in the configuration. These include addresses where the ordering node itself can be found by other components as well as Operations and metrics, which allow you to set up methods for monitoring the health and performance of your ordering node through the configuration of endpoints.

When you're comfortable with how your ordering node has been configured, how your volumes are mounted, and your backend configuration, you can run the command to launch the ordering node (this command will depend on your backend configuration).

Conclusion

Blockchain is highly appraised and endorsed for its decentralized and P2P nature. Blockchain has shown potential in a lot of fields. Due to its property of immutability and the process of consensus involved in processing a transaction, Blockchain has shown its potential for transforming the traditional industry with its key characteristics: decentralization, persistency, anonymity and auditability. In this paper, we present a comprehensive survey on the blockchain. We first give an overview of the blockchain technologies including blockchain architecture and key characteristics of the blockchain. We then discuss the different advantages and working of the Hyperledger Platform. We analyze and compare these protocols in different respects. We also investigate typical blockchain applications. Furthermore, we list some challenges and problems that would hinder blockchain development and summarize some existing approaches for solving these problems. Some possible future directions are also discussed. Nowadays smart contract is developing fast and many smart contract applications are proposed.

References

- [1] Akins, B.W., Chapman, J.L. and Gordon, J.M. (2013) A Whole New World: Income Tax Considerations of the Bitcoin Economy.
- [2] antshares (2016) Antshares Digital Assets for Everyone, <https://www.antshares.org>.
- [3] Atzori, L., Iera, A. and Morabito, G. (2010) ‘The internet of things: a survey’, *Computer Networks*, Vol. 54, No. 15, pp.2787–2805.
- [4] Axon, L. (2015) Privacy-Awareness in Blockchain-based PKI, CDT Technical Paper Series.
- [5] azure (2016) Microsoft Azure: Blockchain as a Service, <https://azure.microsoft.com/enus/solutions/blockchain/>
- [6] Barcelo, J. (2014) User Privacy in the Public Bitcoin Blockchain.
- [7] Bentov, I., Lee, C., Mizrahi, A. and Rosenfeld, M. (2014) ‘Proof of activity: extending Bitcoin’s proof of work via proof of stake [extended abstract]’, *ACM SIGMETRICS Performance Evaluation Review*, Vol. 42, No. 3, pp.34–37. Billah, S. (2015) One Weird Trick to Stop Selfish Miners: Fresh Bitcoins, A Solution for the Honest Miner