SUMMER TRAINING REPORT

On

Implementation of Blockchain using Hyperledger Fabric

Submitted in partial fulfillment of requirements for the award of the

Degree of

Bachelor of Technology

In

Computer Science and Technology

Submitted by

SAATVIK DHURANDHER

06011502721

Under the guidance of

Mr. Shantanu, Scientist E

DRDO, Metcalfe House, New Delhi

Computer Science and Engineering

Bharati Vidyapeeth College of Engineering, New Delhi - 110063, INDIA

August 2023

CERTIFICATE

This is to certify that " Mr. Saatvik Dhurandher " student of BTech in Computer Science and Engineering, from Bharati Vidyapeeth College of Engineering has completed his 5th Semester Training on "Implementation of Blockchain using Hyperledger Fabric" under my supervision at Scientific Analysis Group, Defence Research and Development Organization, New Delhi.

During this tenure we found him sincere and hardworking, we take this opportunity to wish him all the very best in future endeavors.

Training In-charge

Mr. Shantanu

Scientist E

SAG, DRDO

New Delhi

ACKNOWLEDGEMENT

Foremost of all, I express my sincere indebtedness to the “Almighty”, for bestowing me with all the favorable circumstances and kept me in high spirits, I am very grateful to my mentor, **“Mr. Shantanu”** for his patience, encouragement, and many fruitful discussions from the very early stage of this training**.** I wish to express my appreciation to my family for my continuous love and encouragement, for always believing in me, for never failing to provide all the support, and for coping with the pressure that naturally comes with such endeavors.

I am immensely thankful to **“Dr. Deepika Kumar”**, HOD of my department of Computer Science and Engineering for their moral support and continuous inspiration for carrying out the work.

Signature of trainee

Saatvik Dhurandher

Bharati Vidyapeeth College of Engineering

06011502721

B.Tech CSE

5th Sem

TABLE OF CONTENTS

1. INTRODUCTION

Blockchain stands as one of the most dynamic and rapidly advancing technologies globally. A significant 80% of tech organizations are integrating blockchain to address longstanding challenges that have persisted for decades. This technology holds the promise of delivering enduring solutions to sectors including supply chain, payments, logistics, healthcare, digital identity, trade finance, and more. By offering the foundational framework, blockchain facilitates the decentralization and digitization of the mentioned industries.

Imagine you're overseeing a major supermarket chain as its manager. One day, during a casual stroll through one of your stores, you encounter the fruits section. Amidst the neatly arranged rows of fruit, you notice a new item: fresh mango slices, a seasonal delight. Picking up a packet, you're prompted with a question: where did these mangoes come from? Who brought them to market? This curiosity ignites an urge to delve into the mangoes' origin, leading you to enlist your trusted employees to trace their journey. While your employees manage to provide the necessary information, the process takes a grueling seven days, involving extensive paper checks and numerous phone calls.

Now, let's add a twist to the story. Picture customers who purchased the mangoes from your store suddenly voicing complaints about stomach discomfort after consuming them. Seeking the source of the issue becomes more urgent. What once was a casual inquiry now transforms into an emergency search. The seven-day search now carries tangible consequences – potentially legal complications – for both you and your customers. The root cause of this chaos lies in the fact that the information you need is scattered across a complex and diverse supply chain network, making the search as daunting as it sounds.

Suppose you had the ability to trace the mangoes' journey back to the farm, covering every touchpoint along the way. In such a scenario, you could take swift action to prevent fraudulent events in real time. This realization underscores that mangoes traverse various hands on their journey from "farm to forks." However, your only option is to manually sift through documents and restart the tracing process from scratch – a frustrating setback.

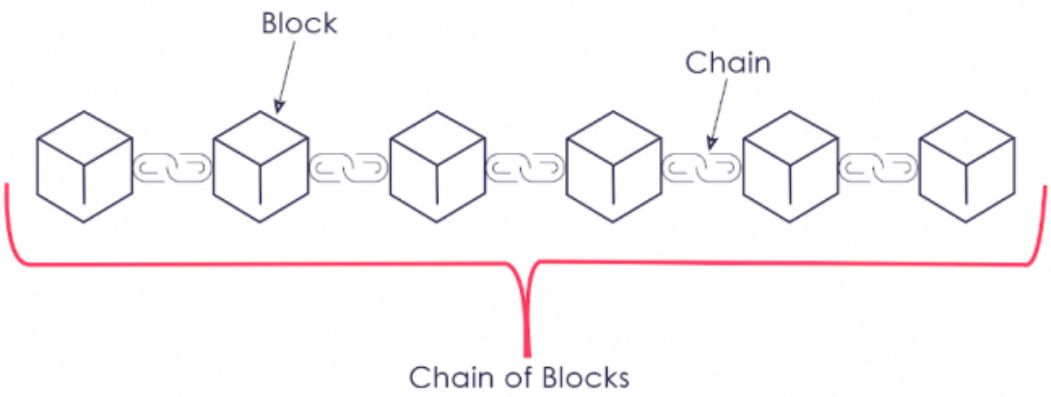
Here comes the picture of Blockchain, that can be used to solve this problem in no time. If we create a blockchain for the selling and add a block every time a transaction of mangoes takes place, we will be able to backtrack the information in a more efficient and faster way.

2. THEORY

Blockchain represents a decentralized and widely distributed ledger capable of storing data across multiple locations without reliance on a central monitoring entity. This technology empowers individuals and businesses to securely store and swiftly exchange both information and currency. In the realm of blockchain, data transmission occurs directly between peers, eliminating the need for intermediaries or middlemen.

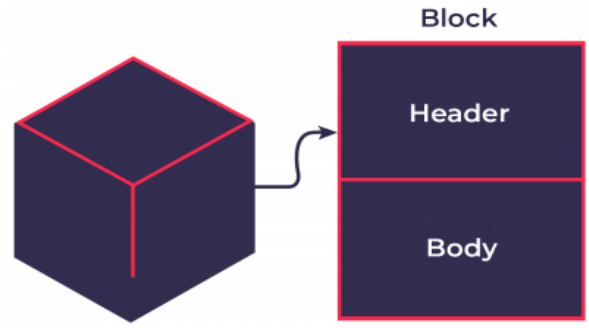
To understand it using daily life examples, in our online interactions, we rely on powerful computers referred to as servers, which are owned by various companies. These servers play a crucial role in facilitating our communication. Similarly, when it comes to financial transactions, we place our trust in banks that oversee the verification and validation of our transactions. These entities, including servers, institutions, and companies, function as intermediaries within our communications and financial dealings. Blockchain technology disrupts this traditional model by removing these intermediaries and introducing a novel approach to communication and transactions.

The term "blockchain" derives from its data storage structure. Within the blockchain framework, data is organized into units known as "blocks." These blocks subsequently interconnect to create a sequential chain alongside other informational blocks, resulting in what we refer to as a "blockchain."



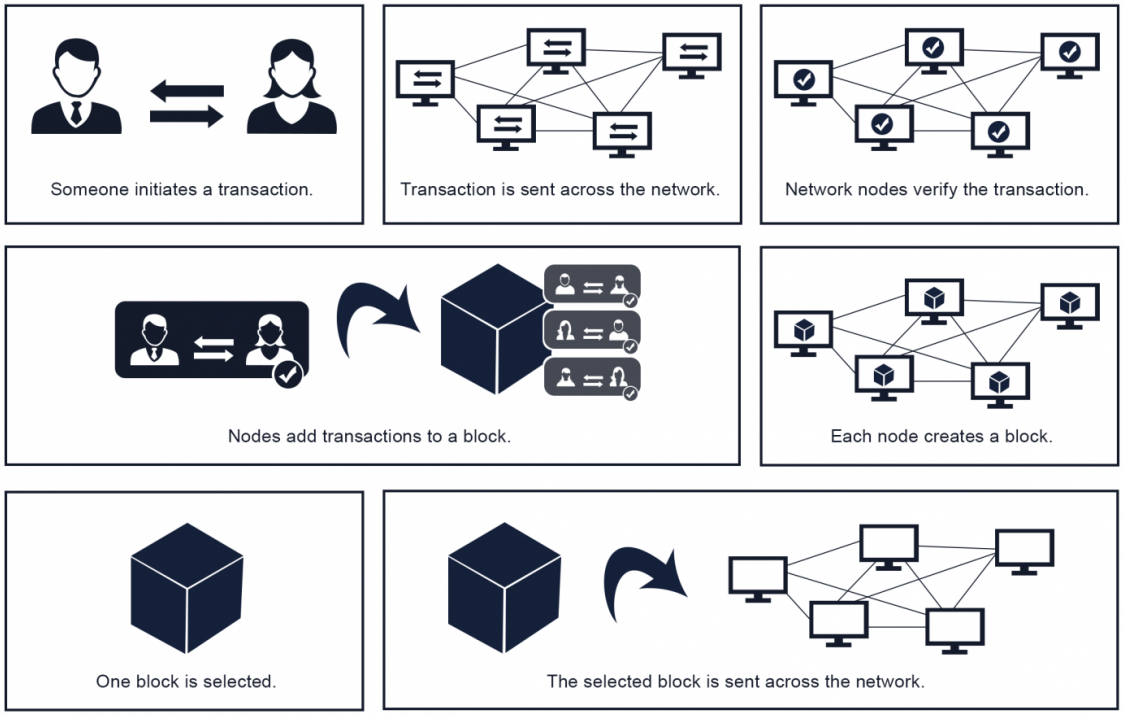
Each block has two main components: a header and a body. Within the body, transactions are stored, while additional information is housed in the header. These blocks undergo cryptographic

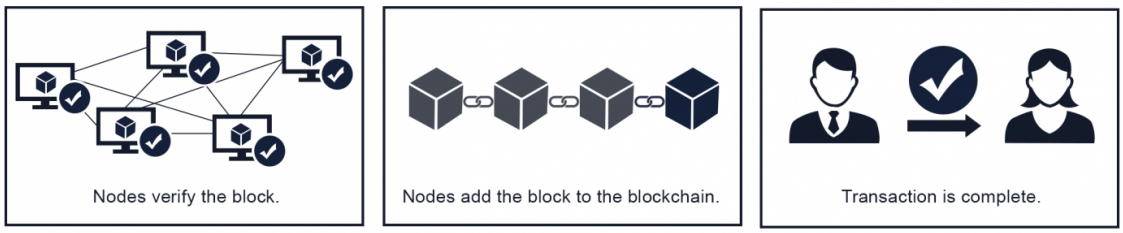
validation and are then linked together to establish an unalterable sequence of blocks known as the blockchain.



2.1 Working Of Blockchain

Blockchain operates in a decentralized manner, relying on numerous computers (nodes) to uphold its records. Whenever a user initiates a transaction within the Blockchain network, the transaction request is disseminated across the peer-to-peer network. Subsequently, the network's nodes validate the transaction's legitimacy. Once confirmed, the transaction joins other verified transactions to form a new block. This block is then safeguarded using cryptographic principles. Each node within the network contributes to creating blocks, and through a selection process, one block is chosen and appended to the existing Blockchain. This step verifies and ensures its security and immutability.





The journey of each blockchain transaction follows these stages:

1. The transaction is broadcasted across the blockchain network.
2. Verification is conducted to validate the transaction's authenticity.
3. Validated transactions are aggregated into a block.
4. From the array of blocks, one is selected.
5. The chosen block is integrated into the chain.

2.2 Types Of Blockchain

Blockchains can be categorized according to their degree of decentralization and accessibility:

Public Blockchain: Open to all for participation and engagement. Activities are transparent and visible to everyone. Examples include Bitcoin and Ethereum.

Private Blockchain: Governed by a single entity, typically employed when network participants are well-known to each other.

Consortium Blockchain: Blending attributes of both public and private blockchains. While not necessarily open to the public, multiple entities form the network. Access to the network is controlled and limited.

2.3 Public Blockchain: Bitcoin

2.3.1 What is Bitcoin ?

Bitcoin serves as both a technological innovation and a currency. Operating on a decentralized global payment network, it operates independently of traditional financial institutions and governmental control.

Pioneering the utilization of blockchain technology, Bitcoin was unveiled by Satoshi Nakamoto in 2008 through the publication of the seminal paper "Bitcoin: A Peer-to-Peer Electronic Cash System." This paper introduced Bitcoin as a digital representation of value, facilitating seamless digital transactions. Functioning as a medium of exchange, unit of account, and store of value, Bitcoin's role encompasses various financial aspects.

In contemporary times, Bitcoin finds widespread utility in the exchange of goods and services. Moreover, it has gained substantial prominence as an investment vehicle, attracting substantial attention from investors seeking diverse opportunities for financial growth.

2.3.2 Components of Bitcoin Network

Bitcoin is composed of four essential components that come together to form its ecosystem.

(1) Software: At its core, Bitcoin operates as a software application. This software utilizes cryptography to facilitate secure transactions and regulate the transfer of bitcoin between parties. Additionally, it manages the creation of new units of bitcoin.

(2) Cryptography: Cryptography plays a pivotal role in the Bitcoin network. It involves the practice and study of techniques aimed at ensuring secure communication in the face of adversarial behavior. This cryptographic foundation underpins the integrity and security of bitcoin transactions.

(3) Hardware: Bitcoin requires hardware resources to operate effectively, much like any other software application. The global Bitcoin network consists of numerous computers, referred to as nodes, distributed across various locations. These nodes collectively participate in processing transactions. There are different types of nodes based on their functionalities:

- Full Node: Maintains the complete blockchain and independently verifies transactions.
- Lightweight Node: Stores only block header information, conserving storage space.
- Miner Node: Engages in blockchain mining using advanced hardware and consensus algorithms. Validates transactions and generates new blocks.
- Router Node: Directs transaction requests to appropriate devices for processing, serving as a network bridge.

Among these nodes, miner nodes have a significant role. They are responsible for validating transactions and creating new blocks in the Bitcoin network.

(4) Miners: The term "miner" is synonymous with its traditional meaning, signifying an entity engaged in extracting valuable resources. In the context of Bitcoin, miners play a vital role in maintaining the network's integrity. To create a new block in the blockchain, miners must solve a computationally challenging cryptographic puzzle. When they successfully solve this puzzle, they are rewarded for their effort with newly minted bitcoin. This incentive system motivates miners to participate and contribute to the network's stability, akin to how gold miners are rewarded with gold for their endeavors.

2.3.3 Working of Bitcoin Network

The total supply of bitcoin is capped at 21 million units, a deliberate measure taken to imbue it with anti-inflationary attributes. The rationale behind this limit is to safeguard the value of bitcoin over time. As miners forge new blocks, new bitcoins are introduced into circulation. Yet, this issuance ceases once the 21 million cap is attained, ensuring a finite supply.

Miners, as integral participants, are rewarded for their role in creating new blocks. This reward comprises transaction fees connected with each transaction. The initial reward bestowed upon blockchain miners was 50 bitcoins. However, this reward undergoes a halving process approximately every four years. At present, each miner receives 6.25 bitcoins as their reward.

In the intricate process of a bitcoin transaction, the transfer of bitcoin from one address to another takes place. To validate the transaction, the sender must affix their signature. This signed transaction traverses the network, undergoing validation by every mining node. The miners then consolidate received transactions to forge a block, encompassing multiple transactions. As a block's size is restricted to 1 MB within the bitcoin blockchain, miners accommodate transactions up to this limit and carry over any excess transactions to the ensuing block.

Upon block creation, miners embark on a competitive puzzle-solving pursuit. When a miner successfully unravels the puzzle, the corresponding block is disseminated to other nodes. These nodes verify the block's authenticity before incorporating it into their blockchain copies. This process ensures uniformity across the network, granting each node a congruent version of the blockchain.

2.4 Private Blockchain: Hyperledger Fabric

2.4.1 Need for Permissioned Blockchain

Public Blockchains like Bitcoin, Etherium, etc. are open to all and anyone can take part in the consensus. But, sometimes it is necessary to restrict the access of certain information to a specific number of people. Let us see some cases:

- Business Alignment: Enterprises often seek exclusive interaction with their trading counterparts when it comes to accessing and updating trade-centric data. In this context, blockchain offers the ability to confine data access to trusted partners, enabling businesses to harness distributed ledgers while safeguarding sensitive information from public exposure.
- Enhanced Transaction Speed: Notorious for their sluggish transaction processing, blockchains such as Bitcoin and Ethereum might not meet the performance demands of businesses reliant on efficient IT infrastructures. Certain permissioned blockchains, on the other hand, boast transaction speeds that can reach several thousand transactions per second (TPS), rendering them an attractive option for numerous enterprises seeking better throughput.
- Structured Governance: Permissioned blockchains cultivate a transparent governance framework among network participants. Consequently,

ensuring the network's progressive evolution is less challenging within permissioned setups.

- Cost-Efficiency: Unlike their public counterparts that involve resource-intensive mining processes, permissioned blockchains adopt algorithms that are less computationally demanding, thereby minimizing costs.

Therefore, there comes the need for a Permissioned blockchain. Some of the key permissioned blockchains are R3 Corda and different Hyperledger projects.

2.4.2 Hyperledger Projects

The Hyperledger Foundation stands as a collaborative endeavor rooted in open-source principles, dedicated to enhancing cross-industry innovations in the realm of blockchain technology. Hosted by the Linux Foundation, this global cooperative boasts the backing of influential leaders in sectors like Banking, Finance, Supply Chains, Internet of Things, Manufacturing, and Technology. Functioning as a nurturing ecosystem, the Hyperledger Foundation fosters novel concepts, extends vital resources, and disseminates outcomes on a global scale. Its pivotal role lies in fostering interoperability, sustainability, transparency, and support, thereby advancing the commercial adoption of blockchain technologies.

Hyperledger Distributed Ledgers:

Hyperledger Distributed Ledgers serve as the cornerstone for creating bespoke blockchain networks. These include:

- Hyperledger Besu
- Hyperledger Fabric
- Hyperledger Indy
- Hyperledger Iroha
- Hyperledger Sawtooth

Hyperledger Tools:

Encompassing a range of functions such as integration and benchmarking, Hyperledger tools bolster the functionalities of various Hyperledger frameworks. Examples of these tools are:

- Hyperledger AnonCreds
- Hyperledger Bevel
- Hyperledger Cacti
- Hyperledger Caliper
- Hyperledger Cello
- Hyperledger Firefly
- Hyperledger Solang

Hyperledger Libraries:

Dedicated to easing development efforts across frameworks, both within and outside of Hyperledger projects, Hyperledger libraries offer reusable toolkits for diverse application development needs. These libraries comprise:

- Hyperledger Ursa
- Hyperledger Transact
- Hyperledger Aries

2.4.3 Hyperledger Fabric Network Components

Peers:

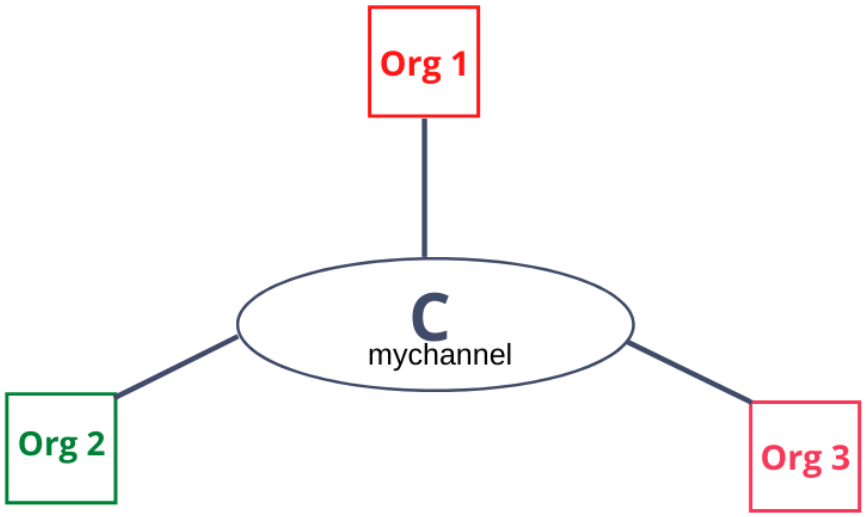
Peers serve as the hosts for ledger instances and smart contracts. Organizational data is stored within the respective peer instances, alongside the deployment and instantiation of smart contracts that encapsulate business functions. Peers can function as endorsing peers or committing peers. In their endorsing role, peers validate transaction proposals from clients, while in their committing role, they incorporate blocks into their blockchain.

Two types of peers exist within the Fabric network:

- Leader Peer
- Anchor Peer

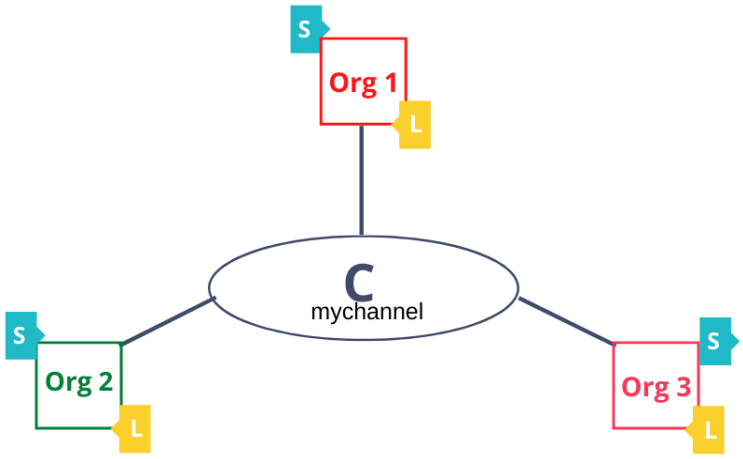
Channels:

Channels facilitate communication among peers, orderers, and applications. Multiple channels can be established within a network, with participants potentially overlapping or differing. Each channel operates under its own rules, ensuring private transactions among participants.



Chaincode:

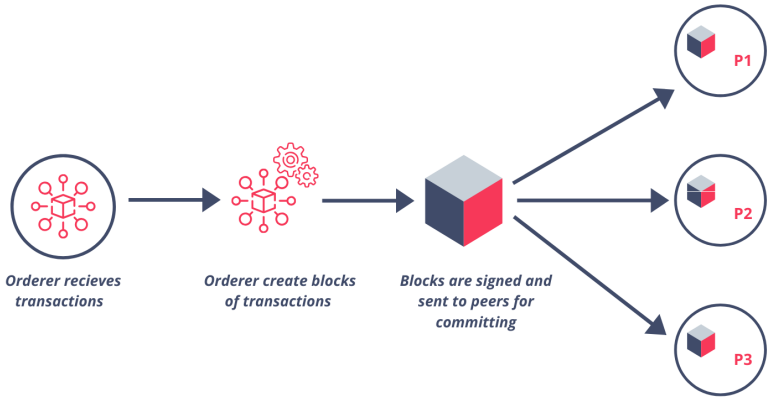
Chaincode encapsulates smart contracts tailored to organizational needs. Hosted within peers, chaincode packages contain the business logic required for operations.



Orderer:

An orderer represents a cluster of nodes responsible for sequencing transactions into blocks and disseminating them to peers. As a network

cornerstone, it constructs and distributes blocks to peers, updating the ledger. Three types of ordering services exist: Solo, Kafka, and Raft. Solo and Kafka were deprecated starting from Hyperledger Fabric version 2.2.



Client:

Applications connect to peers to execute transactions. Built using available SDKs, applications primarily focus on ledger updates and queries. They achieve this by invoking chaincode functions within peers.

Ledger:

The ledger houses actual business asset data. Clients engage in read and write operations on the ledger using CRUD operations, and historical asset values are stored here. Hosting occurs within a peer.

Collaborative Workflow:

The client initiates the transaction proposal to peers, where endorsing peers validate and return the endorsed transaction. Subsequently, the client submits the endorsed transaction to the orderer. Orderers then assemble transaction blocks, distributing them to peers for ledger finalization and updates.

3. Practical Implementation

3.1 Installing Prerequisites

To prepare for the development of Fabric smart contracts, the following software components must be downloaded and set up for Hyperledger Fabric. These installations can be executed on Windows, Linux, or Mac platforms. We will guide you through the installation process for Git, cURL, Node.js, npm, Docker, Docker Compose, and the Fabric installation script.

Git

Git is essential for cloning the fabric-samples repository from GitHub to your local machine. If Git is not already installed, you can obtain it from https://git-scm.com/downloads. After installation, confirm by running the following command:

```
$ git --version
```

cURL

cURL is used to retrieve the Fabric binaries from the web. You can download cURL from https://curl.haxx.se/download.html. After downloading and installing, confirm the installation by running:

```
$ curl -V
```

Node.js and npm

For developing Fabric smart contracts with JavaScript, Node.js and npm are required for processing smart contracts. Supported Node.js versions are 10.15.3 and higher, as well as 12.13.1 and higher. The supported npm version is 6 and higher. Node.js includes npm in its installation. Download Node.js from https://nodejs.org/en/download/. Verify installations with the following commands:

```
$ node -v
$ npm -v
```

Docker and Docker Compose

Hyperledger Fabric's components operate as separate executable services, with Docker images of each component hosted on Docker Hub. A minimum Docker version of 17.06.2-ce is required. Download the latest Docker version from

https://www.docker.com/get-started. Docker Compose is included with Docker. Check your Docker version with:

```
$ docker -v
$ docker-compose -v
```

Ensure Docker is running before proceeding, as it's necessary for the Fabric installation script.

Fabric Installation Script

Navigate to the directory where you intend to install the Fabric binaries and sample projects. Make sure Docker is running, as the script requires Docker to download the Fabric images. The script will accomplish the following:

- Download Fabric binaries
- Clone fabric-samples from the GitHub repository
- Download Hyperledger Fabric Docker images

Execute the script using the command:

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

By following these steps, you will have the necessary software components ready to start developing Fabric smart contracts.

3.2 Developing a Smart Contract

Chaincode, written in Go, Node.js, or Java, serves as a program that adheres to a defined interface. It operates in a separate process from the peer, overseeing the initiation and management of the ledger state through transactions initiated by applications. Functioning similar to a "smart contract," chaincode primarily manages the agreed-upon business logic among network members. Through proposal transactions, a chaincode can be invoked for ledger updates or queries. With the necessary authorization, a chaincode holds the ability to trigger another chaincode, whether in the same channel or across different channels, to access its state. Notably, if the invoked chaincode exists on a distinct channel from the invoking one, it's limited to read queries. In such cases, the invoked chaincode on a different channel solely executes a Query function, which does not partake in state validation checks during subsequent commit phases.

3.2.1 Asset Transfer Chaincode

Our application is a basic sample chaincode to initialize a ledger with documents as assets. It helps to create, read, update, and delete documents, check to see if a document exists, and transfer documents from one owner to another.

First, let us create a Struct “Document” to represent the document as an asset in the ledger.

```
const Document = {
            ID: id,
            TimeOfOrigin: timeOfOrigin,
            ModificationDate: modificationDate,
            Author: author,
            label: label,
          };
```

Next, let us implement the “InitLedger” function to populate the ledger with some initial data.

```
async InitLedger(ctx) {
  const Documents = [
      {
          ID: 'Document1',
          TimeOfOrigin: '12:24:48',
          ModificationDate: '12-5-2023',
          Author: 'Ram',
          Label: 'Confidential',
      },
      {
          ID: 'Document2',
          TimeOfOrigin: '14:45:32',
          ModificationDate: '4-9-2022',
          Author: 'Shyam',
          Label: 'Confidential',
      },
      {
          ID: 'Document3',
          TimeOfOrigin: '17:32:51',
```

```
                ModificationDate: '10-7-2022',
                Author: 'Raghav',
                Label: 'Public',
            },
            {
                ID: 'Document4',
                TimeOfOrigin: '20:21:22',
                ModificationDate: '2-5-2021',
                Author: 'Manav',
                Label: 'Public',
            },
            {
                ID: 'Document5',
                TimeOfOrigin: '7:12:13',
                ModificationDate: '15-8-2023',
                Author: 'Adi',
                Label: 'Public',
            },
            {
                ID: 'Document6',
                TimeOfOrigin: '3:22:22',
                ModificationDate: '17-4-2020',
                Author: 'Raman',
                Label: 'Confidential',
            },
        ];

        for (const Document of Documents) {
            Document.docType = 'Document';
            await ctx.stub.putState(Document.ID,
            Buffer.from(stringify(sortKeysRecursive(Document))));
        }
    }
```

Next, we write a function to create a document on the ledger that does not yet exist. When writing chaincode, we check for the existence of something on the ledger prior to taking an action on it, as is demonstrated in the “CreateDocument” function below.

```
async CreateDocument(ctx, id, timeOfOrigin, modificationDate,
author, label) {
const exists = await this.DocumentExists(ctx, id);
```

```
        if (exists) {
         throw new Error(`The Document ${id} already exists`);
          }

    const Document = {
              ID: id,
              TimeOfOrigin: timeOfOrigin,
              ModificationDate: modificationDate,
              Author: author,
              label: label,
            };
    await ctx.stub.putState(id,
    Buffer.from(stringify(sortKeysRecursive(Document))));
          return JSON.stringify(Document);
        }
```

Now that we have populated the ledger with some initial documents and created a document, let's write a function "ReadDocument" that allows us to read a document from the ledger.

```
    async ReadDocument(ctx, id) {
      const DocumentJSON = await ctx.stub.getState(id);
      if (!DocumentJSON || DocumentJSON.length === 0) {
          throw new Error(`The Document ${id} does not exist`);
      }
      return DocumentJSON.toString();
  }
```

Now that we have documents on our ledger we can interact with, let's write a chaincode function "UpdateDocument" that allows us to update attributes of the document that we are allowed to change.

```
    async UpdateDocument(ctx, id, timeOfOrigin, modificationDate,
    author, label) {
    const exists = await this.DocumentExists(ctx, id);
      if (!exists) {
          throw new Error(`The Document ${id} does not exist`);
      }
    const updatedDocument = {
          ID: id,
          Author: author,
          TimeOfOrigin: timeOfOrigin,
          ModificationDate: modificationDate,
```

```
            Label: label,
        };
    return ctx.stub.putState(id,
    Buffer.from(stringify(sortKeysRecursive(updatedDocument))));}
```

There may be cases where we need the ability to delete a document from the ledger, so let's write a "DeleteDocument" function to handle that requirement.

```
    async DeleteDocument(ctx, id) {
        const exists = await this.DocumentExists(ctx, id);
        if (!exists) {
            throw new Error(`The Document ${id} does not exist`);
        }
        return ctx.stub.deleteState(id);
    }
```

It is a good practice to check whether a document already exists in the ledger or not. So, we will write a "DocumentExists" function to do that for us.

```
    async DocumentExists(ctx, id) {
        const DocumentJSON = await ctx.stub.getState(id);
        return DocumentJSON && DocumentJSON.length > 0;
    }
```

Next, we'll write a function we'll call "TransferDocument" that enables the transfer of a document from one owner to another.

```
    async TransferDocument(ctx, id, newauthor) {
        const DocumentString = await this.ReadDocument(ctx, id);
        const Document = JSON.parse(DocumentString);
        const oldauthor = Document.author;
        Document.author = newauthor;
    await ctx.stub.putState(id,
    Buffer.from(stringify(sortKeysRecursive(Document))));
        return oldauthor;
    }
```

Let's write a function we'll call "GetAllDocuments" that enables the querying of the ledger to return all of the documents on the ledger.

```
    async GetAllDocuments(ctx) {
        const allResults = [];
        const iterator = await ctx.stub.getStateByRange('', '');
        let result = await iterator.next();
        while (!result.done) {
```

```
            const strValue =
  Buffer.from(result.value.value.toString()).toString('utf8');
            let record;
            try {
                record = JSON.parse(strValue);
            } catch (err) {
                console.log(err);
                record = strValue;
            }
            allResults.push(record);
            result = await iterator.next();
        }
        return JSON.stringify(allResults);
}
```

4. Observation

Now, we deploy this chaincode on the test network to see whether it works or not.

For doing this, first we will start the test network,

```
saatvikd@DESKTOP-V0NGKH0:~/go/src/github.com/fabric-samples/test-network$ ./network.sh up
Using docker and docker-compose
Starting nodes with CLI timeout of '5' tries and CLI delay of '3' seconds and using database 'leveldb' with crypto from 'cryptogen'
LOCAL_VERSION=v2.5.3
DOCKER_IMAGE_VERSION=v2.5.3
/home/saatvikd/go/src/github.com/fabric-samples/test-network/../bin/cryptogen
Generating certificates using cryptogen tool
Creating Org1 Identities
+ cryptogen generate --config=./organizations/cryptogen/crypto-config-org1.yaml --output=organizations
org1.example.com
+ res=0
Creating Org2 Identities
+ cryptogen generate --config=./organizations/cryptogen/crypto-config-org2.yaml --output=organizations
org2.example.com
+ res=0
Creating Orderer Org Identities
+ cryptogen generate --config=./organizations/cryptogen/crypto-config-orderer.yaml --output=organizations
+ res=0
Generating CCP files for Org1 and Org2
[+] Running 8/8
 ✔ Network fabric_test                      Created                                                                  0.1s
 ✔ Volume "compose_orderer.example.com"     Created                                                                  0.0s
 ✔ Volume "compose_peer0.org1.example.com"  Created                                                                  0.0s
 ✔ Volume "compose_peer0.org2.example.com"  Created                                                                  0.0s
 ✔ Container peer0.org2.example.com         Started                                                                  1.4s
 ✔ Container orderer.example.com            Started                                                                  1.4s
 ✔ Container peer0.org1.example.com         Started                                                                  1.4s
 ✔ Container cli                            Started                                                                  1.8s
```

Then, we will create a channel named ‘mychannel’

```
saatvikd@DESKTOP-V0NGKH0:~/go/src/github.com/fabric-samples/test-network$ ./network.sh createChannel
Using docker and docker-compose
Creating channel 'mychannel'.
If network is not up, starting nodes with CLI timeout of '5' tries and CLI delay of '3' seconds and using database 'leveldb
Network Running Already
Using docker and docker-compose
Generating channel genesis block 'mychannel.block'
/home/saatvikd/go/src/github.com/fabric-samples/test-network/../bin/configtxgen
+ configtxgen -profile TwoOrgsApplicationGenesis -outputBlock ./channel-artifacts/mychannel.block -channelID mychannel
2023-08-26 23:39:15.618 IST 0001 INFO [common.tools.configtxgen] main -> Loading configuration
2023-08-26 23:39:15.625 IST 0002 INFO [common.tools.configtxgen.localconfig] completeInitialization -> orderer type: etcdraft
2023-08-26 23:39:15.625 IST 0003 INFO [common.tools.configtxgen.localconfig] completeInitialization -> Orderer.EtcdRaft.Options unset
, setting to tick_interval:"500ms" election_tick:10 heartbeat_tick:1 max_inflight_blocks:5 snapshot_interval_size:16777216
2023-08-26 23:39:15.625 IST 0004 INFO [common.tools.configtxgen.localconfig] Load -> Loaded configuration: /home/saatvikd/go/src/gith
ub.com/fabric-samples/test-network/configtx/configtx.yaml
2023-08-26 23:39:15.629 IST 0005 INFO [common.tools.configtxgen] doOutputBlock -> Generating genesis block
2023-08-26 23:39:15.629 IST 0006 INFO [common.tools.configtxgen] doOutputBlock -> Creating application channel genesis block
2023-08-26 23:39:15.630 IST 0007 INFO [common.tools.configtxgen] doOutputBlock -> Writing genesis block
+ res=0
Creating channel mychannel
Using organization 1
+ osnadmin channel join --channelID mychannel --config-block ./channel-artifacts/mychannel.block -o localhost:7053 --ca-file /home/sa
atvikd/go/src/github.com/fabric-samples/test-network/organizations/ordererOrganizations/example.com/tlsca/tlsca.example.com-cert.pem
--client-cert /home/saatvikd/go/src/github.com/fabric-samples/test-network/organizations/ordererOrganizations/example.com/orderers/or
derer.example.com/tls/server.crt --client-key /home/saatvikd/go/src/github.com/fabric-samples/test-network/organizations/ordererOrgan
izations/example.com/orderers/orderer.example.com/tls/server.key
+ res=0
Status: 201
{
        "name": "mychannel",
        "url": "/participation/v1/channels/mychannel",
        "consensusRelation": "consenter",
        "status": "active",
        "height": 1
```

```
ll, '"version":' '"0"' '},' '"Writers":' '{' '"mod_policy":' '"",' '"policy":' null, '"version":' '"0"' '}' '},' '"values":' '{' '"MS
P":' '{' '"mod_policy":' '"",' '"value":' null, '"version":' '"0"' '}' '},' '"version":' '"0"' '}' '},' '"mod_policy":' '"",' '"polic
ies":' '{},' '"values":' '{},' '"version":' '"0"' '}' '},' '"mod_policy":' '"",' '"policies":' '{},' '"values":' '{},' '"version":' '
"0"' '},' '"write_set":' '{' '"groups":' '{' '"Application":' '{' '"groups":' '{' '"Org2MSP":' '{' '"groups":' '{},' '"mod_policy":'
'"Admins",' '"policies":' '{' '"Admins":' '{' '"mod_policy":' '"",' '"policy":' null, '"version":' '"0"' '},' '"Endorsement":' '{' '"
mod_policy":' '"",' '"policy":' null, '"version":' '"0"' '},' '"Readers":' '{' '"mod_policy":' '"",' '"policy":' null, '"version":' '
"0"' '},' '"Writers":' '{' '"mod_policy":' '"",' '"policy":' null, '"version":' '"0"' '}' '},' '"values":' '{' '"AnchorPeers":' '{' '
"mod_policy":' '"Admins",' '"value":' '{' '"anchor_peers":' '[' '{' '"host":' '"peer0.org2.example.com",' '"port":' 9051 '}' ']' '},'
 '"version":' '"0"' '},' '"MSP":' '{' '"mod_policy":' '"",' '"value":' null, '"version":' '"0"' '}' '},' '"version":' '"1"' '}' '},'
'"mod_policy":' '"",' '"policies":' '{},' '"values":' '{},' '"version":' '"0"' '}' '},' '"mod_policy":' '"",' '"policies":' '{},' '"v
alues":' '{},' '"version":' '"0"' '}' '}}}}'
+ configtxlator proto_encode --input config_update_in_envelope.json --type common.Envelope --output Org2MSPanchors.tx
2023-08-26 18:09:26.338 UTC 0001 INFO [channelCmd] InitCmdFactory -> Endorser and orderer connections initialized
2023-08-26 18:09:26.355 UTC 0002 INFO [channelCmd] update -> Successfully submitted channel update
Anchor peer set for org 'Org2MSP' on channel 'mychannel'
Channel 'mychannel' joined
```

Once we have created the channel, we can deploy our chaincode into the network,

```
saatvikd@DESKTOP-V0NGKH0:~/go/src/github.com/fabric-samples/test-network$ ./network.sh deployCC -ccn basic -ccp ../asset-transfer-bas
ic/chaincode-javascript/ -ccl javascript
Using docker and docker-compose
deploying chaincode on channel 'mychannel'
executing with the following
- CHANNEL_NAME: mychannel
- CC_NAME: basic
- CC_SRC_PATH: ../asset-transfer-basic/chaincode-javascript/
- CC_SRC_LANGUAGE: javascript
- CC_VERSION: 1.0
- CC_SEQUENCE: 1
- CC_END_POLICY: NA
- CC_COLL_CONFIG: NA
- CC_INIT_FCN: NA
- DELAY: 3
- MAX_RETRY: 5
- VERBOSE: false
+ peer lifecycle chaincode package basic.tar.gz --path ../asset-transfer-basic/chaincode-javascript/ --lang node --label basic_1.0
+ res=0
++ peer lifecycle chaincode calculatepackageid basic.tar.gz
+ PACKAGE_ID=basic_1.0:feb6f04056662317dfcf800949b205508d88485d3fdc888294e442ffec19264f
Chaincode is packaged
Installing chaincode on peer0.org1...
Using organization 1
+ peer lifecycle chaincode queryinstalled --output json
+ jq -r 'try (.installed_chaincodes[].package_id)'
+ grep '^basic_1.0:feb6f04056662317dfcf800949b205508d88485d3fdc888294e442ffec19264f$'
+ test 1 -ne 0
+ peer lifecycle chaincode install basic.tar.gz
```

```
}
Checking the commit readiness of the chaincode definition successful on peer0.org2 on channel 'mychannel'
Using organization 1
Using organization 2
+ peer lifecycle chaincode commit -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile /home/saatvikd/go
/src/github.com/fabric-samples/test-network/organizations/ordererOrganizations/example.com/tlsca/tlsca.example.com-cert.pem --channel
ID mychannel --name basic --peerAddresses localhost:7051 --tlsRootCertFiles /home/saatvikd/go/src/github.com/fabric-samples/test-netw
ork/organizations/peerOrganizations/org1.example.com/tlsca/tlsca.org1.example.com-cert.pem --peerAddresses localhost:9051 --tlsRootCe
rtFiles /home/saatvikd/go/src/github.com/fabric-samples/test-network/organizations/peerOrganizations/org2.example.com/tlsca/tlsca.org
2.example.com-cert.pem --version 1.0 --sequence 1
+ res=0
2023-08-27 10:43:18.019 IST 0001 INFO [chaincodeCmd] ClientWait -> txid [e532005c1ff5933f16d57932a59c588a76aed2b953996e405fb59d2339c4
e654] committed with status (VALID) at localhost:9051
2023-08-27 10:43:18.022 IST 0002 INFO [chaincodeCmd] ClientWait -> txid [e532005c1ff5933f16d57932a59c588a76aed2b953996e405fb59d2339c4
e654] committed with status (VALID) at localhost:7051
Chaincode definition committed on channel 'mychannel'
Using organization 1
Querying chaincode definition on peer0.org1 on channel 'mychannel'...
Attempting to Query committed status on peer0.org1, Retry after 3 seconds.
+ peer lifecycle chaincode querycommitted --channelID mychannel --name basic
+ res=0
Committed chaincode definition for chaincode 'basic' on channel 'mychannel':
Version: 1.0, Sequence: 1, Endorsement Plugin: escc, Validation Plugin: vscc, Approvals: [Org1MSP: true, Org2MSP: true]
Query chaincode definition successful on peer0.org1 on channel 'mychannel'
Using organization 2
Querying chaincode definition on peer0.org2 on channel 'mychannel'...
Attempting to Query committed status on peer0.org2, Retry after 3 seconds.
+ peer lifecycle chaincode querycommitted --channelID mychannel --name basic
+ res=0
Committed chaincode definition for chaincode 'basic' on channel 'mychannel':
Version: 1.0, Sequence: 1, Endorsement Plugin: escc, Validation Plugin: vscc, Approvals: [Org1MSP: true, Org2MSP: true]
Query chaincode definition successful on peer0.org2 on channel 'mychannel'
Chaincode initialization is not required
```

Now we will use the peer CLI to interact with the network. Peer CLI is present in the bin folder of fabric-samples. So, first we will add the peer binaries to the CLI,

```
saatvikd@DESKTOP-V0NGKH0:~/go/src/github.com/fabric-samples/test-network$ export PATH=${PWD}/../bin:$PATH
saatvikd@DESKTOP-V0NGKH0:~/go/src/github.com/fabric-samples/test-network$ export FABRIC_CFG_PATH=$PWD/../config/
```

And then we will set the environment variables to operate the peer CLI as Org1

```
saatvikd@DESKTOP-V0NGKH0:~/go/src/github.com/fabric-samples/test-network$ export CORE_PEER_TLS_ENABLED=true
export CORE_PEER_LOCALMSPID="Org1MSP"
export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt
export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
export CORE_PEER_ADDRESS=localhost:7051
saatvikd@DESKTOP-V0NGKH0:~/go/src/github.com/fabric-samples/test-network$
```

Now that we have set the environment variables, we will now invoke the function “InitLedger” to load the network with the documents,

```
saatvikd@DESKTOP-V0NGKH0:~/go/src/github.com/fabric-samples/test-network$ peer chaincode invoke -o localhost:7050 --ordererTLSHostnam
eOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/
tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n basic --peerAddresses localhost:7051 --tlsRootCertFiles "${PWD}/organizations/
peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRootCertFiles "${PWD}
/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt" -c '{"function":"InitLedger","Args":[]}'
2023-08-27 11:12:20.203 IST 0001 INFO [chaincodeCmd] chaincodeInvokeOrQuery -> Chaincode invoke successful. result: status:200
```

Let us see the document and their details present in the ledger,

```
saatvikd@DESKTOP-V0NGKH0:~/go/src/github.com/fabric-samples/test-network$ peer chaincode query -C mychannel -n basic -c '{"Args":["Ge
tAllDocuments"]}'
[{"Author":"Ram","ID":"Document1","Label":"Confidential","ModificationDate":"12-5-2023","TimeOfOrigin":"12:24:48","docType":"Document
"},{"Author":"Shyam","ID":"Document2","Label":"Confidential","ModificationDate":"4-9-2022","TimeOfOrigin":"14:45:32","docType":"Docum
ent"},{"Author":"Raghav","ID":"Document3","Label":"Public","ModificationDate":"10-7-2022","TimeOfOrigin":"17:32:51","docType":"Docume
nt"},{"Author":"Manav","ID":"Document4","Label":"Public","ModificationDate":"2-5-2021","TimeOfOrigin":"20:21:22","docType":"Document"
},{"Author":"Adi","ID":"Document5","Label":"Public","ModificationDate":"15-8-2023","TimeOfOrigin":"7:12:13","docType":"Document"},{"A
uthor":"Raman","ID":"Document6","Label":"Confidential","ModificationDate":"17-4-2020","TimeOfOrigin":"3:22:22","docType":"Document"}]
```

Now, we will add a new Document in the ledger,

```
saatvikd@DESKTOP-V0NGKH0:~/go/src/github.com/fabric-samples/test-network$ peer chaincode invoke -o localhost:7050 --ordererTLSHostnam
eOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/
tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n basic --peerAddresses localhost:7051 --tlsRootCertFiles "${PWD}/organizations/
peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRootCertFiles "${PWD}
/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt" -c '{"function":"CreateDocument","Args":["
Document7","12:22:24","27-08-2023","Nabh","Permissioned"]}'
2023-08-27 12:25:27.089 IST 0001 INFO [chaincodeCmd] chaincodeInvokeOrQuery -> Chaincode invoke successful. result: status:200 payloa
d:"{\"ID\":\"Document7\",\"TimeOfOrigin\":\"12:22:24\",\"ModificationDate\":\"27-08-2023\",\"Author\":\"Nabh\",\"label\":\"Permission
ed\"}"
```

Next, let us update the Author of “Document6” from “Raman” to “Madhav”,

```
saatvikd@DESKTOP-V0NGKH0:~/go/src/github.com/fabric-samples/test-network$ peer chaincode invoke -o localhost:7050 --ordererTLSHostnam
eOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/
tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n basic --peerAddresses localhost:7051 --tlsRootCertFiles "${PWD}/organizations/
peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRootCertFiles "${PWD}
/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt" -c '{"function":"TransferDocument","Args":
["Document6","Madhav"]}'
2023-08-27 12:29:10.900 IST 0001 INFO [chaincodeCmd] chaincodeInvokeOrQuery -> Chaincode invoke successful. result: status:200
```

Now say we don’t want “Document3”, so let us delete it,

```
saatvikd@DESKTOP-V0NGKH0:~/go/src/github.com/fabric-samples/test-network$ peer chaincode invoke -o localhost:7050 --ordererTLSHostnam
eOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/
tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n basic --peerAddresses localhost:7051 --tlsRootCertFiles "${PWD}/organizations/
peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRootCertFiles "${PWD}
/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt" -c '{"function":"DeleteDocument","Args":["
Document3"]}'
2023-08-27 12:32:36.802 IST 0001 INFO [chaincodeCmd] chaincodeInvokeOrQuery -> Chaincode invoke successful. result: status:200 payloa
d:"{\"type\":\"Buffer\",\"data\":[]}"
```

Let us see the updated Ledger,

```
saatvikd@DESKTOP-V0NGKH0:~/go/src/github.com/fabric-samples/test-network$ peer chaincode query -C mychannel -n basic -c '{"Args":["Ge
tAllDocuments"]}'
[{"Author":"Ram","ID":"Document1","Label":"Confidential","ModificationDate":"12-5-2023","TimeOfOrigin":"12:24:48","docType":"Document
"},{"Author":"Shyam","ID":"Document2","Label":"Confidential","ModificationDate":"4-9-2022","TimeOfOrigin":"14:45:32","docType":"Docum
ent"},{"Author":"Manav","ID":"Document4","Label":"Public","ModificationDate":"2-5-2021","TimeOfOrigin":"20:21:22","docType":"Document
"},{"Author":"Adi","ID":"Document5","Label":"Public","ModificationDate":"15-8-2023","TimeOfOrigin":"7:12:13","docType":"Document"},{"
Author":"Raman","ID":"Document6","Label":"Confidential","ModificationDate":"17-4-2020","TimeOfOrigin":"3:22:22","author":"Madhav","do
cType":"Document"},{"Author":"Nabh","ID":"Document7","ModificationDate":"27-08-2023","TimeOfOrigin":"12:22:24","label":"Permissioned"
}]
```

Now that we have done working with the ledger, let us bring down the network,

```
saatvikd@DESKTOP-V0NGKH0:~/go/src/github.com/fabric-samples/test-network$ ./network.sh down
Using docker and docker-compose
Stopping network
[+] Running 9/9
 ✔ Container orderer.example.com          Removed                                                                    0.7s
 ✔ Container cli                          Removed                                                                    0.6s
 ✔ Container peer0.org2.example.com       Removed                                                                    1.5s
 ✔ Container peer0.org1.example.com       Removed                                                                    1.4s
 ✔ Volume compose_orderer.example.com     Removed                                                                    0.0s
 ✔ Volume compose_peer0.org1.example.com  Removed                                                                    0.0s
 ✔ Volume compose_peer0.org2.example.com  Removed                                                                    0.0s
 ✔ Volume compose_peer0.org3.example.com  Removed                                                                    0.0s
 ✔ Network fabric_test                    Removed                                                                    0.3s
Error response from daemon: get docker_orderer.example.com: no such volume
Error response from daemon: get docker_peer0.org1.example.com: no such volume
Error response from daemon: get docker_peer0.org2.example.com: no such volume
Removing remaining containers
Removing generated chaincode docker images
```

5. Conclusion

In conclusion, the implementation of a Hyperledger Fabric-based blockchain network marks a significant stride towards transforming industries through decentralized and secure transactions. The journey from understanding the foundational principles of blockchain technology to the practical deployment of Hyperledger Fabric has been enlightening. This report has explored the various components of the Fabric network, delving into its architecture, nodes, channels, and the pivotal role of chaincode.

By adopting Hyperledger Fabric, organizations gain access to a robust framework that supports business consortia, supply chain transparency, and secure data management. The integration of smart contracts through chaincode empowers users to automate processes and ensure trust among network participants. The report has highlighted the nuances of deploying and configuring a Fabric network, emphasizing the importance of software prerequisites, Docker images, and the installation script.

It is evident that Hyperledger Fabric provides a versatile platform for businesses to collaboratively build applications, streamline operations, and drive innovation. The enhanced security, scalability, and customizable features offered by Fabric underscore its potential to reshape industries across diverse sectors. As we conclude this report, the journey of implementing Hyperledger Fabric serves as a testament to the promising future of blockchain technology, inspiring us to explore more use cases and advance towards a decentralized and transparent digital world.

REFERENCES

1. https://learn.kba.ai/course/blockchain-foundation-program
2. https://learn.kba.ai/course/hyperledger-fabric-fundamentals
3. https://github.com/hyperledger/fabric-samples
4. Matt Zand, Xun (Brian) Wu, Mark Anthony Morris, Hands-On Smart Contract Development with Hyperledger Fabric V2_ Building Enterprise Blockchain Applications: O'Reilly Media (2021)