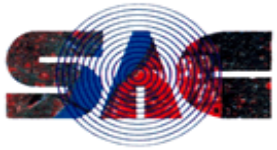


Sensitive Data Access Control using private Blockchain Network

Report submitted as part
of
Internship at Space Applications Center

Submitted by
SARTHAK AGGARWAL
(Registration No. RS01172)
B.Tech. (CSE with Specialization in Blockchain Technology)

Under the guidance of

Gulshan Gupta
Sci/Eng – SD
ITND/CSIG/MISA
Space Applications Centre, (ISRO) Ahmedabad

Institution
Vellore Institute of Technology
Vellore – 632014
Tamil Nadu

SRTD-RTCG-MISA
Space Applications Centre (ISRO)
Ahmedabad, Gujarat

05 September 2023 to 30 November 2023

TABLE OF CONTENTS

Acknowledgements

I would like to express my heartfelt gratitude to the Space Application Centre (SAC), Indian Space Research Organisation (ISRO) for granting me the incredible opportunity to intern at this prestigious institution. This internship has been a transformative experience, and I am truly appreciative of the chance to contribute to the remarkable work being carried out at SAC.

I would like to extend my special thanks to my guide, Mr. Gulshan Gupta, for his unwavering support, guidance, and mentorship throughout the course of my internship. His expertise, patience, and willingness to share knowledge have been invaluable in shaping my understanding of the Blockchain technology field. Mr. Gupta's dedication to my growth and development as an intern has left a lasting impact on me, and I am truly fortunate to have had such a knowledgeable and caring mentor.

I also want to thank the entire team at SAC for their warm welcome, cooperation, and encouragement. The learning and exposure I received during my time at SAC have been instrumental in my personal and professional growth, and I will carry these experiences with me as I continue my journey in the field of space technology.

I am deeply grateful for the trust and confidence that SAC and Mr. Gulshan Gupta placed in me, and I look forward to applying the knowledge and skills I have acquired during this internship in my future endeavors. This experience has been a significant milestone in my academic and career development, and I am truly thankful for this wonderful opportunity.

Sincerely,

Sarthak Aggarwal

About Space Applications Centre (SAC), Indian Space and Research Organization (ISRO)

- The *Indian Space Research Organisation* (ISRO) has long been recognized as a pioneer in the field of space technology and exploration. Established in 1969, ISRO has consistently achieved remarkable milestones in satellite technology, space missions, and scientific advancements. Its endeavors have not only enhanced India's prestige in the international space community but have also contributed significantly to various sectors, including communication, agriculture, and weather forecasting.
- *Space Applications Centre* (SAC) is one of the major centers of the Indian Space Research Organisation (ISRO). SAC focuses on the design of space-borne instruments for ISRO missions and development and operationalisation of applications of space technology for societal benefits. The applications cover communication, broadcasting, navigation, disaster monitoring, meteorology, oceanography, environment monitoring and natural resources survey.

The Space Application Centre (SAC) in Ahmedabad, a distinguished wing of the Indian Space Research Organisation (ISRO), stands as a beacon of technological excellence in the field of space applications. Located in Ahmedabad, SAC plays a pivotal role in the development and execution of satellite systems and space-based applications. With a mission to harness space technology for national development, SAC is at the forefront of satellite design, fabrication, and application development.

My internship at SAC, ISRO, Ahmedabad, has been an immersive experience within this dynamic and visionary organization. Engaging with state-of-the-art projects and collaborating with experts has provided invaluable insights into the intricacies of space technology. This report encapsulates the essence of my internship journey, showcasing the impactful work conducted at SAC, Ahmedabad, and the invaluable contributions made to India's space exploration endeavors.

CHAPTER - 1
Introduction

Objective and Scope of the Project

The core focus of the internship is building an **'Sensitive Data Access Control using private Blockchain Network".** For the Course of this project we will be using a framework developed by Hyperledger known as Hyperledger Fabric. Hyperledger Fabric is an open-source blockchain framework developed to provide a modular and scalable platform for developing enterprise-grade blockchain applications. This technology choice aligns with ISRO's requirement of handling highly sensitive data with no single point of compromise.

Internship Timeline

This internship at SAC, ISRO spanned for a period of three months, from September 5, 2023, to November 30, 2023. During this time, I collaborated with ISRO's experienced professionals, gaining invaluable insights into the space industry and blockchain technology. This hands-on experience not only enhanced my academic knowledge but also provided a platform to contribute to ISRO's mission and goals.

Importance of Blockchain Technology for ISRO

Using permissioned blockchains can offer several benefits to organizations like the Indian Space Research Organisation (ISRO) when implementing blockchain technology in their operations. Here are some key advantages of using permissioned blockchains for organizations like ISRO:

- **Data Security:** Blockchain ensures the security and immutability of data, critical for the confidentiality and integrity of ISRO's sensitive information.
- **Supply Chain Management:** Blockchain can streamline the complex supply chain of space missions, ensuring transparency and traceability of components.
- **Smart Contracts:** The utilization of smart contracts can automate various processes within ISRO, reducing administrative overhead.

- **Auditing and Compliance:** Blockchain's transparent ledger can simplify auditing and compliance processes for regulatory authorities.
- **Research and Development:** The project contributes to the ongoing research and development efforts in integrating blockchain technology with space exploration and satellite technology.

In summary, permissioned blockchains offer ISRO a secure, efficient, and compliant way to manage data, streamline operations, and collaborate with trusted partners. By leveraging blockchain technology, ISRO can enhance data integrity, security, and transparency while meeting the unique challenges and requirements of space exploration and research.

CHAPTER - 2
Literature Review

Blockchain Technology

Blockchain is a decentralized immutable append-only public ledger. Blockchain securely records and verifies transactions across a network of computers. It consists of a chain of blocks, each containing a set of transactions, linked together in chronological order. What sets blockchain apart is its immutability and transparency—once information is added to a block, it becomes extremely difficult to alter, providing a high level of data integrity. This technology has found applications far beyond cryptocurrencies, including supply chain management, voting systems, and smart contracts, offering a new paradigm for trust and data management in various industries.

Evolution of Blockchain Technology

The evolution of blockchain technology represents a transformative journey from its conceptual inception to its current state as a foundational pillar of the digital age. Emerging in 2008 with the enigmatic Satoshi Nakamoto's whitepaper on Bitcoin, blockchain was initially conceived as a decentralized ledger for cryptocurrencies. However, its potential soon transcended digital cash, giving rise to a myriad of innovative applications across diverse sectors. Over the years, blockchain has evolved through distinct phases, including the emergence of altcoins, the advent of smart contracts through platforms like Ethereum, and the development of enterprise-grade solutions such as Hyperledger. Its impact extends beyond finance to supply chains, healthcare, voting systems, and more, heralding a paradigm shift in how trust, transparency, and security are established in the digital realm. This evolutionary journey underscores the significance of blockchain as a technological powerhouse reshaping industries and challenging traditional norms.

Introduction to Public and Private Blockchains

- **Public Blockchain**

  Public blockchains are decentralized networks where anyone can participate, validate transactions, and become a part of the consensus mechanism. They are open and permissionless, meaning that anyone can join, read, and write data on the blockchain without needing explicit authorization.

- **Private Blockchain**

  Private blockchains are permissioned networks where access and participation are restricted to a select group of participants who have explicit permissions to read, write, and validate transactions. These networks are often used within organizations or consortiums for specific business or governance purposes.

Key Characteristics of a Blockchain Network

- **Decentralization:** In a blockchain network, there is no central authority or intermediary. Instead, data is stored and verified across a network of nodes (computers) that work together to reach consensus on the validity of transactions.
- **Distributed Ledger:** Transactions are recorded in a ledger that is distributed to and synchronized across all nodes in the network. This distributed ledger ensures that all participants have access to the same data, reducing the risk of a single point of failure.
- **Immutability:** Once a transaction is added to the blockchain, it becomes extremely difficult to alter or delete. This immutability is achieved through cryptographic hashing and consensus mechanisms.
- **Transparency:** The data in the blockchain is transparent and visible to all participants in the network. Each participant can independently verify transactions.
- **Security:** Blockchain uses cryptographic techniques to secure transactions and control access. Transactions are cryptographically signed, and data is stored in blocks that are linked together using cryptographic hashes.
- **Consensus Mechanisms:** To add a new block of transactions to the blockchain, nodes in the network must reach consensus. Various consensus mechanisms, such

as Proof of Work (PoW) and Proof of Stake (PoS), are used to validate and agree on the contents of the blockchain.

Applications of Blockchain Technology

Blockchain technology has applications across various industries, including finance, supply chain management, healthcare, voting systems, and more. Some common use cases include:

- **Cryptocurrencies:** As mentioned, cryptocurrencies like Bitcoin and Ethereum are built on blockchain technology.
- **Supply Chain Management:** Blockchain can track the origin and movement of products in supply chains, enhancing transparency and traceability.
- **Identity Verification:** Blockchain can be used to securely verify and manage identities, reducing the risk of identity theft.
- **Smart Contracts:** These self-executing contracts automate and enforce agreements in a transparent and tamper-proof manner.
- **Cross-Border Payments:** Blockchain can simplify and expedite cross-border transactions.
- **Voting Systems:** Blockchain-based voting systems aim to enhance the security and transparency of elections.
- **Healthcare:** Patient records and medical data can be securely stored and shared among authorized parties using blockchain.

Overall, blockchain technology offers a new paradigm for securely and transparently recording and verifying transactions and data, potentially disrupting various industries and business processes.

Permissioned Blockchain Network

A private blockchain, also known as a permissioned blockchain, is a type of blockchain network where access and participation are restricted to known and trusted entities. Unlike public blockchains, where anyone can join and participate, private blockchains operate with controlled access and specific permissions. Here's a detailed explanation of private/permissioned blockchains and the reasons to use them:

Key Characteristics

- **Restricted Access:** Only authorized participants are allowed to join the network. Access is controlled through an access control list (ACL) or by invitation. Participants are typically identified and verified entities, such as organizations, businesses, or individuals.
- **Identity Verification:** Participants on a private blockchain are required to undergo identity verification, ensuring that each participant is known and trusted. This contrasts with public blockchains, where participants can remain pseudonymous.
- **Consensus Mechanisms:** Private blockchains often use more efficient consensus mechanisms than the energy-intensive Proof of Work (PoW) used in public blockchains like Bitcoin. Common consensus mechanisms for private blockchains include Practical Byzantine Fault Tolerance (PBFT), Raft, or delegated proof of stake (DPoS).
- **Transaction Privacy:** Transaction privacy and confidentiality are prioritized in private blockchains. While transactions are recorded on the blockchain, they are typically encrypted or made visible only to authorized parties. This is essential for protecting sensitive business or personal data.
- **Governance:** Governance in private blockchains is often more straightforward and adaptable. Participants have more control over network rules, updates, and decisions, making it easier to reach consensus on changes.

Advantages

- **Privacy and Confidentiality:** Private blockchains are ideal for scenarios where data privacy is paramount. Businesses may need to protect sensitive information, such as financial records or customer data, while still benefiting from blockchain technology.
- **Regulatory Compliance:** Some industries, like finance and healthcare, are subject to strict regulations and compliance requirements. Private blockchains allow organizations to implement the necessary controls and adhere to regulatory guidelines while still leveraging blockchain benefits.
- **Efficiency and Scalability:** Private blockchains can be optimized for high transaction throughput and low latency, making them suitable for enterprise-level

applications. This can significantly improve efficiency compared to public blockchains.

- **Trust Among Participants:** In business consortia or supply chain networks, participants often know and trust each other. A permissioned blockchain formalizes this trust, streamlining transactions and reducing the need for intermediaries.
- **Control and Governance:** Organizations can maintain more control over network governance and decision-making in private blockchains. They can adapt the blockchain to their specific needs and adjust rules as necessary.
- **Reduced Energy Consumption:** By using more efficient consensus mechanisms, private blockchains consume less energy compared to the energy-intensive PoW used in public blockchains like Bitcoin and Ethereum.
- **Customization:** Private blockchains can be customized to meet specific business requirements, including features like multi-signature transactions, asset tokenization, and customized smart contracts.

Frameworks for Private Blockchain Implementation

Implementing a private blockchain network involves choosing the right framework or platform to build and manage the network. Several blockchain frameworks and platforms are available for this purpose. Here are some commonly used ones:

- **Hyperledger Fabric:** Hyperledger Fabric is an open-source blockchain framework hosted by the Linux Foundation. It is designed for enterprise use cases and offers features like permissioned networks, modular architecture, and support for smart contracts. Fabric allows organizations to create private, permissioned networks with fine-grained access control.
- **Ethereum (Private Fork):** Ethereum, a public blockchain, can be forked to create a private Ethereum-based network. This allows organizations to leverage Ethereum's smart contract capabilities while maintaining control over access and data privacy.
- **Corda:** Corda is an open-source blockchain platform developed by R3 for building private, permissioned blockchain networks. It is designed for financial services and focuses on privacy and scalability.

- **Quorum:** Quorum is an open-source blockchain platform developed by JPMorgan Chase. It is based on Ethereum but offers enhanced privacy features suitable for enterprise use cases.
- **BESU (formerly Pantheon):** BESU is an open-source Ethereum client developed by ConsenSys. It supports both public and private Ethereum networks and can be used to create permissioned networks.
- **Multichain:** Multichain is a platform for creating and deploying private blockchains. It offers simplicity and a set of APIs for building custom blockchain applications.

For the course of the internship, we will be focusing majorly on the implementation of a private blockchain network using the framework provided by *Hyperledger Fabric.* Hyperledger Fabric is renowned globally for its open-source, permissioned blockchain framework. Its versatility, scalability, and robust security features make it a top choice for enterprises seeking to build decentralized applications. Its modular architecture, consensus mechanisms, and active developer community further contribute to its widespread adoption in various industries.

Background Study

Non-Fungible Tokens (NFTs)

NFTs, or Non-Fungible Tokens, are a type of digital asset that represents ownership or proof of authenticity of a unique item, piece of content, or piece of data. Unlike cryptocurrencies like Bitcoin or Ethereum, which are fungible and can be exchanged on a one-to-one basis, NFTs are unique and indivisible. They are typically used to represent ownership of digital or physical assets, including digital art, collectibles, music, virtual real estate, and more.

Access Control for NFTs:

Access control for NFTs involves mechanisms for authorizing and authenticating users to perform actions related to NFTs. Here's how it typically works:

Authentication:

- Ownership Proof: When a user claims ownership of an NFT, they need to prove that they possess the private key associated with the NFT's public key. This is typically done by signing a message with the private key, demonstrating that they are the legitimate owner.
- User Identity: The NFT owner's identity can be linked to their public key using a Public Key Infrastructure (PKI) approach. The user's digital certificate and public key are associated with their identity.

Authorization:

- Access Control Lists (ACL): ACLs are commonly used to manage authorization for NFTs. An ACL is a list of permissions associated with users or user roles. It specifies who can perform certain actions on an NFT, such as transferring it, minting new NFTs, or accessing associated content.
- Role-Based Access Control (RBAC): NFT platforms may implement RBAC to control access. Roles such as owner, creator, curator, and viewer can be defined, each with specific permissions. For example, creators may be allowed to mint NFTs, while viewers can only view them.
- Smart Contracts: In some cases, NFTs are associated with smart contracts that contain access control logic. The smart contract enforces rules and permissions for NFTs. When a user attempts an action, the smart contract verifies their permissions based on the ACL or RBAC rules.

Use of ACL in NFTs:

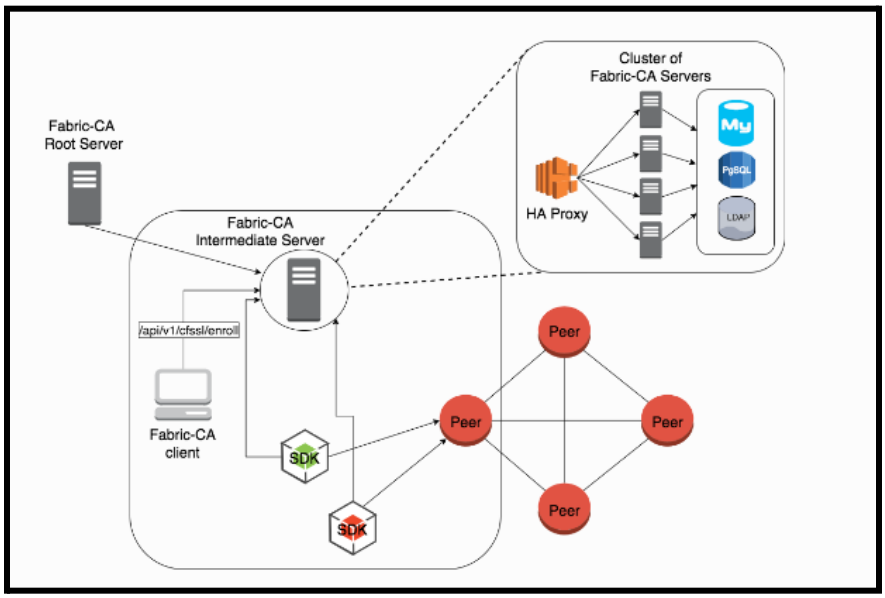
- Content Access: ACLs can be used to control access to the content associated with an NFT. For example, an NFT representing digital art may grant viewing access only to specific users or roles specified in the ACL.
- Transfer Authorization: To prevent unauthorized transfers of NFTs, ACLs can specify who has the authority to transfer ownership. Users not listed in the ACL may not be able to transfer the NFT.
- Minting and Creation: ACLs can determine who has the privilege to mint new NFTs. Creators and artists can be granted minting privileges while other users are not.

- Editions and Collections: ACLs can be used to manage editions and collections of NFTs. For example, only certain users or roles may be able to create new editions or collections.

Access control mechanisms, including ACLs and smart contracts, help ensure that NFTs are used and accessed in a way that aligns with the rights and permissions defined by the NFT owners and creators. They play a crucial role in maintaining the security, authenticity, and value of NFTs in various applications, including art, gaming, and digital collectibles. We will be using a similar kind of access control mechanism for our network.

Fabric Certificate Authority (CA)

CAs are so important that Fabric provides a built-in CA component to allow you to create CAs in the blockchain networks you form. This component — known as Fabric CA is a private root CA provider capable of managing digital identities of Fabric participants that have the form of X.509 certificates. Because Fabric CA is a custom CA targeting the Root CA needs of Fabric, it is inherently not capable of providing SSL certificates for general/automatic use in browsers. However, because some CA must be used to manage identity (even in a test environment), Fabric CA can be used to provide and manage certificates. It is also possible — and fully appropriate — to use a public/commercial root or intermediate CA to provide identification.



Docker

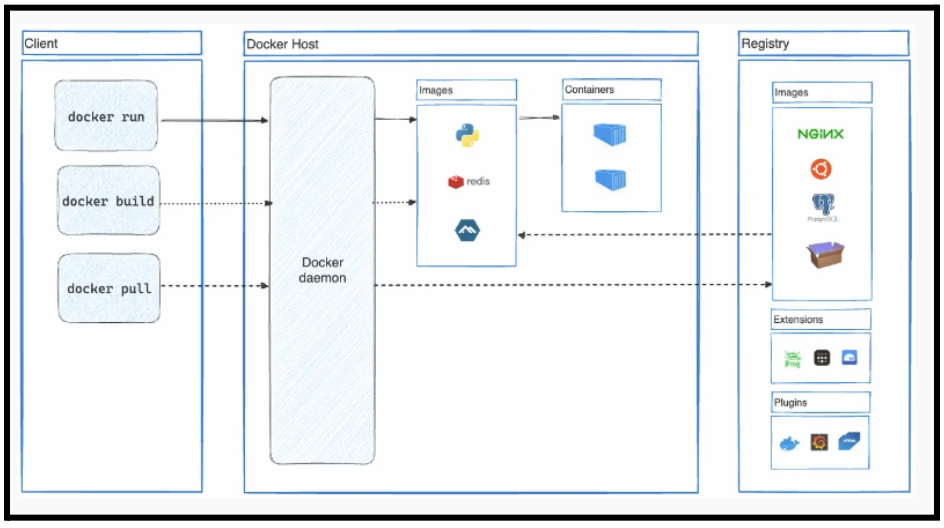
Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security lets you run many containers simultaneously on a given host. Containers are lightweight and contain everything needed to run the application, so you don't need to rely on what's installed on the host. You can share containers while you work, and be sure that everyone you share with gets the same container that works in the same way.

Docker provides tooling and a platform to manage the lifecycle of your containers:

- Develop your application and its supporting components using containers.
- The container becomes the unit for distributing and testing your application.

Docker Architecture

Docker uses a client-server architecture. The Docker client talks to the Docker daemon, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface. Another Docker client is Docker Compose, that lets you work with applications consisting of a set of containers.



CHAPTER - 3
Hyperledger Fabric

Overview

Hyperledger Fabric is a cutting-edge, open-source blockchain framework that has redefined the way industries approach decentralized ledger technology. Developed under the Linux Foundation's Hyperledger project, Fabric stands out as a premier choice for businesses and organizations seeking to harness the power of blockchain for their specific needs. With its focus on permissioned networks, modular architecture, and robust smart contract capabilities, Hyperledger Fabric empowers enterprises to establish secure and highly customizable private blockchain networks. Its impact extends across various sectors, revolutionizing processes, enhancing security, and ensuring transparency in a rapidly evolving digital landscape.

Advantages Over Other Frameworks like Ethereum

- **Permissioned Model:** Hyperledger Fabric is purpose-built for permissioned networks, ensuring that only authorized participants can join, enhancing security and confidentiality. This is especially advantageous for enterprises with strict access control requirements.
- **Privacy and Confidentiality:** Fabric offers sophisticated privacy features such as private channels and data isolation, allowing sensitive information to remain confidential among involved parties. Ethereum, by design, has limited privacy.
- **Modular Architecture:** The modular architecture of Fabric enables organizations to customize their blockchain networks to suit specific business needs. This flexibility is crucial for tailoring the network to meet unique requirements.
- **Pluggable Consensus:** Fabric supports pluggable consensus mechanisms, allowing organizations to select the most appropriate consensus algorithm for their use case. Ethereum's consensus mechanisms are less versatile.

- **Scalability:** Fabric's Practical Byzantine Fault Tolerance (PBFT) consensus mechanism and private channels contribute to higher transaction throughput and scalability, making it well-suited for enterprise-level applications.
- **Enterprise Adoption:** Hyperledger Fabric has gained significant traction in enterprise adoption due to its focus on privacy, modularity, and strong industry support.
- **Regulatory Compliance:** Fabric's design facilitates regulatory compliance, making it attractive to industries with stringent regulatory requirements.
- **Versatile Programming Languages:** Fabric supports multiple programming languages, allowing a broader range of developers to participate in smart contract development.

Architecture and Components

Hyperledger Fabric is a versatile, permissioned blockchain framework known for its modular architecture and scalability. It's designed for enterprise use cases and offers flexibility in terms of consensus, privacy, and programming languages. Here's an overview of its key architectural components:

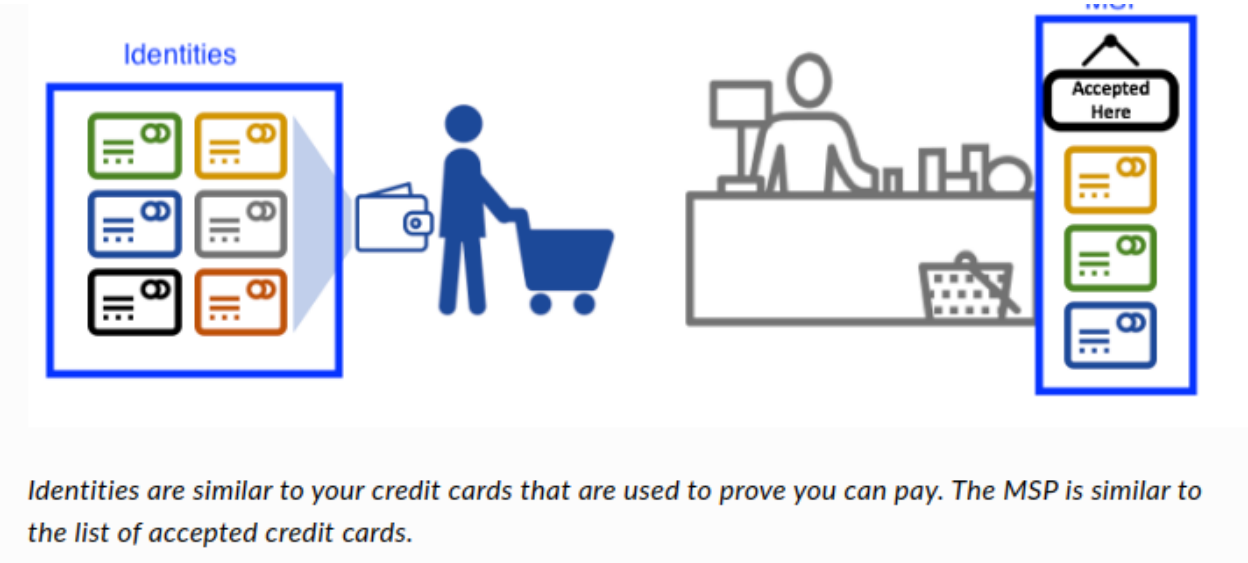
- ## Membership Service Provider (MSP):

  Despite its name, the Membership Service Provider does not actually provide anything. Rather, the implementation of the MSP requirement is a set of folders that are added to the configuration of the network and is used to define an organization both inwardly (organizations decide who its admins are) and outwardly (by allowing other organizations to validate that entities have the authority to do what they are attempting to do). Whereas Certificate Authorities generate the certificates that represent identities, the MSP contains a list of permissioned identities.

  The MSP identifies which Root CAs and Intermediate CAs are accepted to define the members of a trust domain by listing the identities of their members, or by identifying which CAs are authorized to issue valid identities for their members.

  Because Fabric is a permissioned network, blockchain participants need a way to prove their identity to the rest of the network in order to transact on the network.

Certificate Authorities issue identities by generating a public and private key which forms a key-pair that can be used to prove identity. This identity needs a way to be recognized by the network, which is where the MSP comes in. For example, a peer uses its private key to digitally sign, or endorse, a transaction. The MSP is used to check that the peer is allowed to endorse the transaction. The public key from the peer's certificate is then used to verify that the signature attached to the transaction is valid. Thus, the MSP is the mechanism that allows that identity to be trusted and recognized by the rest of the network.



Identities are similar to your credit cards that are used to prove you can pay. The MSP is similar to the list of accepted credit cards.

MSPs occur in two domains in a blockchain network:

- Locally on an actor's node (local MSP)
- In channel configuration (channel MSP)

The key difference between local and channel MSPs is not how they function – both turn identities into roles – but their scope. Each MSP lists roles and permissions at a particular level of administration.

Local MSPs

Local MSPs are defined for clients and for nodes (peers and orderers). Local MSPs define the permissions for a node (who are the peer admins who can operate the node, for example). The local MSPs of clients allow the user to authenticate itself in

its transactions as a member of a channel (e.g. in chaincode transactions), or as the owner of a specific role into the system such as an organization admin, for example, in configuration transactions.

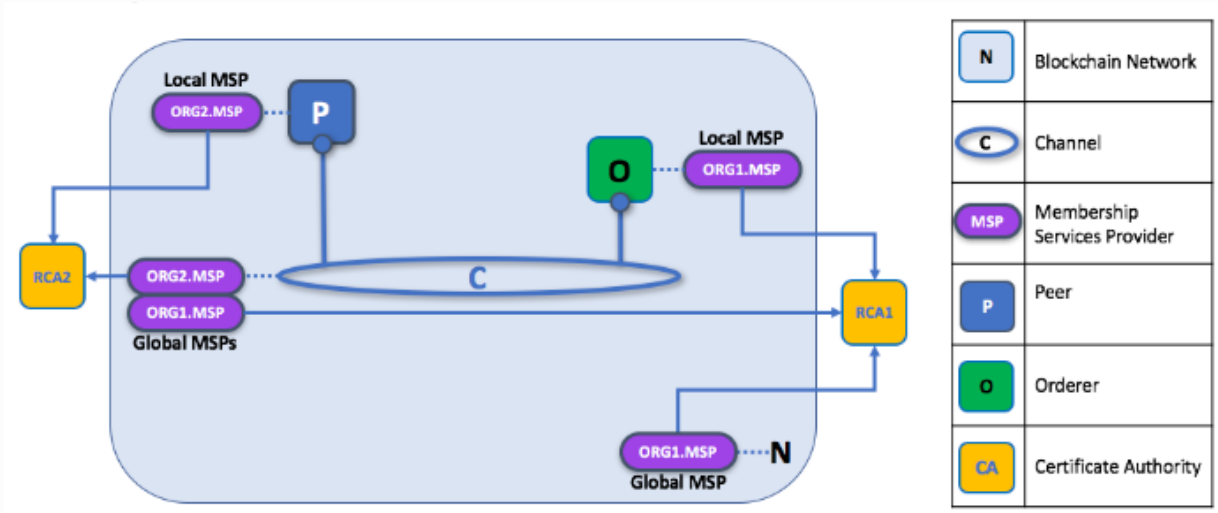
Every node must have a local MSP defined, as it defines who has administrative or participatory rights at that level (peer admins will not necessarily be channel admins, and vice versa). This allows for authenticating member messages outside the context of a channel and to define the permissions over a particular node (who has the ability to install chaincode on a peer, for example).

Channel MSPs

Channel MSPs define administrative and participatory rights at the channel level. Peers and ordering nodes on an application channel share the same view of channel MSPs, and will therefore be able to correctly authenticate the channel participants. This means that if an organization wishes to join the channel, an MSP incorporating the chain of trust for the organization's members would need to be included in the channel configuration. Otherwise transactions originating from this organization's identities will be rejected. Whereas local MSPs are represented as a folder structure on the file system, channel MSPs are described in a channel configuration.

Channel MSPs identify who has authority at a channel level. The channel MSP defines the relationship between the identities of channel members (which themselves are MSPs) and the enforcement of channel level policies. Channel MSPs contain the MSPs of the organizations of the channel members. Every organization participating in a channel must have an MSP defined for it.

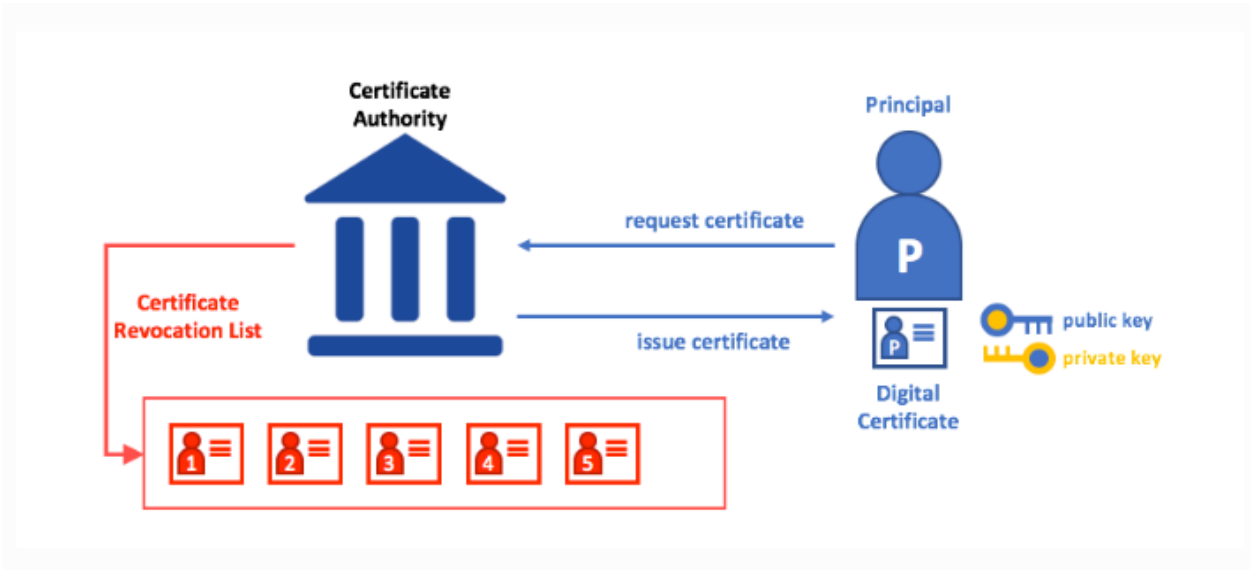
The following diagram illustrates how local and channel MSPs coexist on the network:



- **Identity:**

  The different actors in a blockchain network include peers, orderers, client applications, administrators and more. Each of these actors — active elements inside or outside a network able to consume services — has a digital identity encapsulated in an X.509 digital certificate. These identities really matter because they determine the exact permissions over resources and access to information that actors have in a blockchain network.

  A public key infrastructure (PKI) is a collection of internet technologies that provides secure communications in a network. The elements of Public Key Infrastructure (PKI). A PKI is composed of Certificate Authorities who issue digital certificates to parties (e.g., users of a service, service provider), who then use them to authenticate themselves in the messages they exchange in their environment. A CA's Certificate Revocation List (CRL) constitutes a reference for the certificates that are no longer valid. Revocation of a certificate can happen for a number of reasons



There are four key elements to PKI:

- Digital Certificates
- Public and Private Keys
- Certificate Authorities
- Certificate Revocation Lists

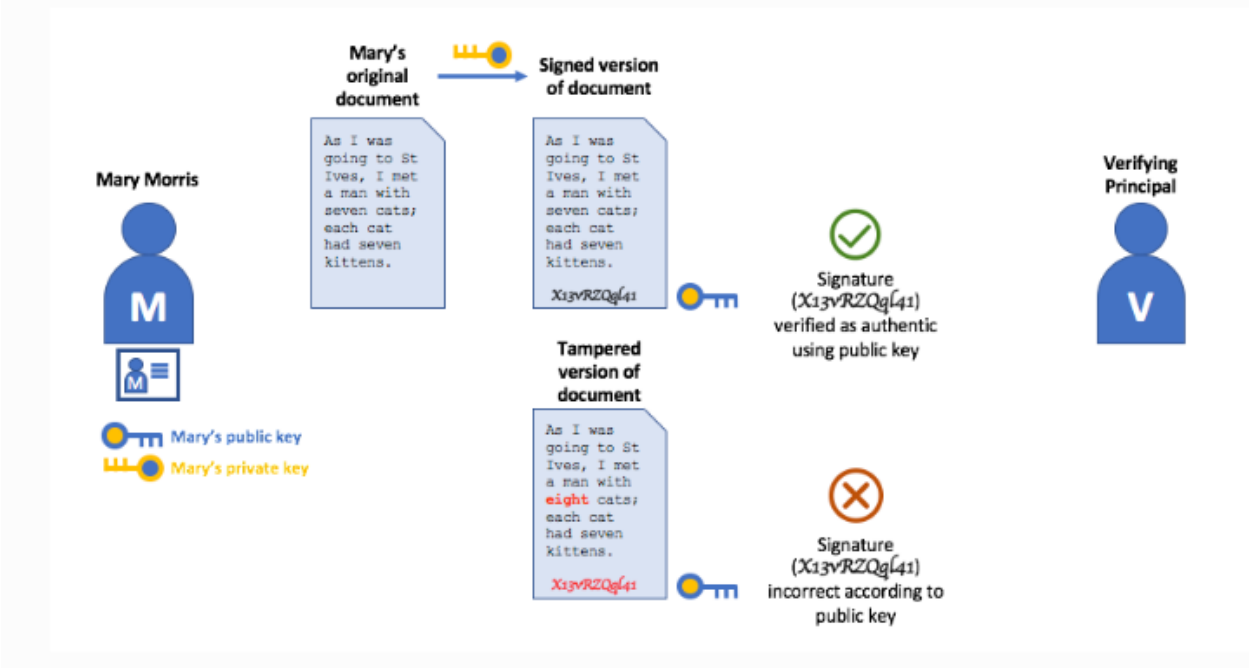
Digital Certificates

A digital certificate is a document which holds a set of attributes relating to the holder of the certificate. The most common type of certificate is the one compliant with the X.509 standard, which allows the encoding of a party's identifying details in its structure.

Public and Private Keys

Technically speaking, digital signature mechanisms require each party to hold two cryptographically connected keys: a public key that is made widely available and acts as authentication anchor, and a private key that is used to produce digital signatures on messages. Recipients of digitally signed messages can verify the origin and integrity of a received message by checking that the attached signature is valid under the public key of the expected sender.

The unique relationship between a private key and the respective public key is the cryptographic magic that makes secure communications possible. The unique mathematical relationship between the keys is such that the private key can be used to produce a signature on a message that only the corresponding public key can match, and only on the same message.

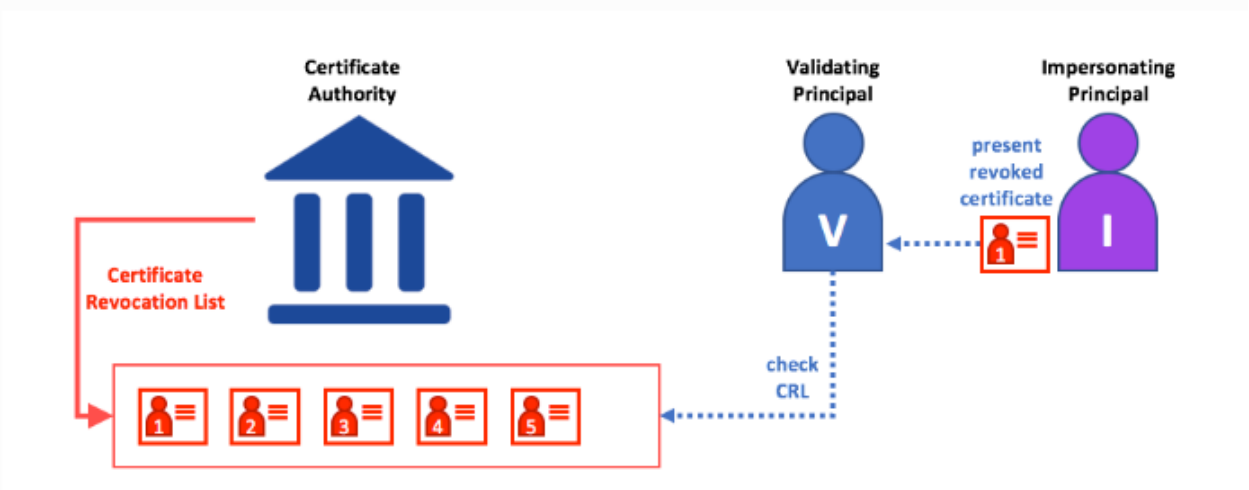


Certificate Authorities

A Certificate Authority dispenses certificates to different actors. These certificates are digitally signed by the CA and bind together the actor with the actor's public key (and optionally with a comprehensive list of properties). As a result, if one trusts the CA (and knows its public key), it can trust that the specific actor is bound to the public key included in the certificate, and owns the included attributes, by validating the CA's signature on the actor's certificate.

Certificate Revocation Lists

A Certificate Revocation List (CRL) is easy to understand — it's just a list of references to certificates that a CA knows to be revoked for one reason or another. When a third party wants to verify another party's identity, it first checks the issuing CA's CRL to make sure that the certificate has not been revoked. A verifier doesn't have to check the CRL, but if they don't they run the risk of accepting a compromised identity.



- **Ordering Service:**

  The Ordering Service maintains the order of transactions in the blockchain network. It ensures that all nodes in the network agree on the order of transactions before

they are committed to the ledger. Hyperledger Fabric supports various consensus algorithms for ordering, including *Kafka*-based ordering and Solo (for testing).

Orderers and the transaction flow

Phase one: Transaction Proposal and Endorsement

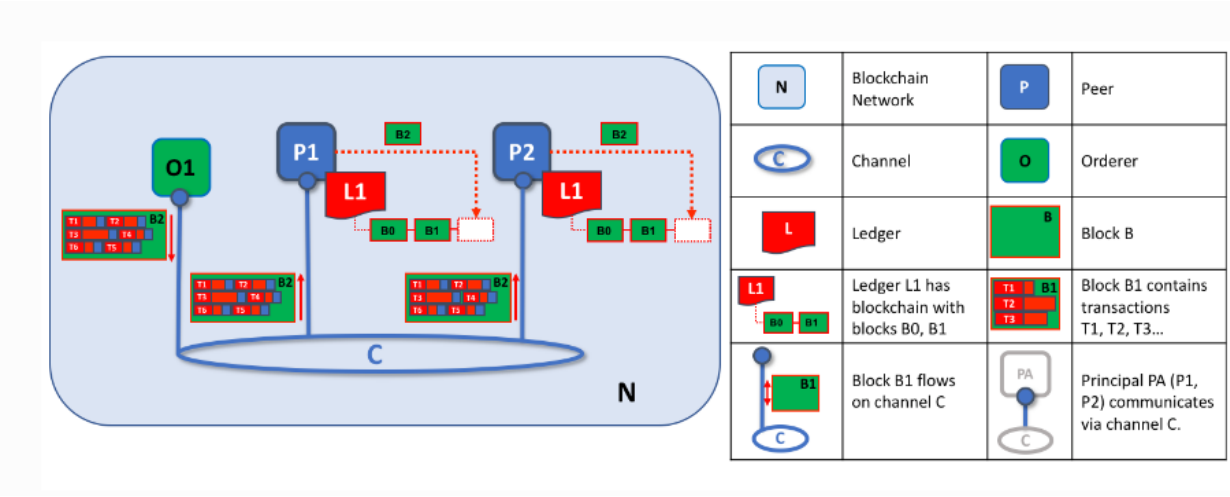
In the first phase, a client application sends a transaction proposal to the Fabric Gateway service, via a trusted peer. This peer executes the proposed transaction or forwards it to another peer in its organization for execution.

Phase two: Transaction Submission and Ordering

For an endorsed transaction, the gateway service forwards the transaction to the ordering service, which orders it with other endorsed transactions, and packages them all into a block. The ordering service creates these blocks of transactions, which will ultimately be distributed to all peers on the channel for validation and commitment to the ledger in phase three. The blocks themselves are also ordered and are the basic components of a blockchain ledger.

Phase three: Transaction Validation and Commitment

The third phase of the transaction workflow involves the distribution of ordered and packaged blocks from the ordering service to the channel peers for validation and commitment to the ledger. Phase three begins with the ordering service distributing blocks to all channel peers. It's worth noting that not every peer needs to be connected to an orderer — peers can cascade blocks to other peers using the gossip protocol — although receiving blocks directly from the ordering service is recommended. Each peer will validate distributed blocks independently, ensuring that ledgers remain consistent.



Ordering service implementations

Raft

Raft is a crash fault tolerant (CFT) ordering service based on an implementation of Raft protocol etc. Raft follows a "leader and follower" model, where a leader node is elected (per channel) and its decisions are replicated by the followers. Raft ordering services should be easier to set up and manage than Kafka-based ordering services, and their design allows different organizations to contribute nodes to a distributed ordering service.

Kafka

Similar to Raft-based ordering, Apache Kafka is a CFT implementation that uses a "leader and follower" node configuration. Kafka utilizes a ZooKeeper ensemble for management purposes. The Kafka based ordering service has been available since Fabric v1.0, but many users may find the additional administrative overhead of managing a Kafka cluster intimidating or undesirable.

Solo

The Solo implementation of the ordering service is intended for test only and consists only of a single ordering node. It has been deprecated and may be removed

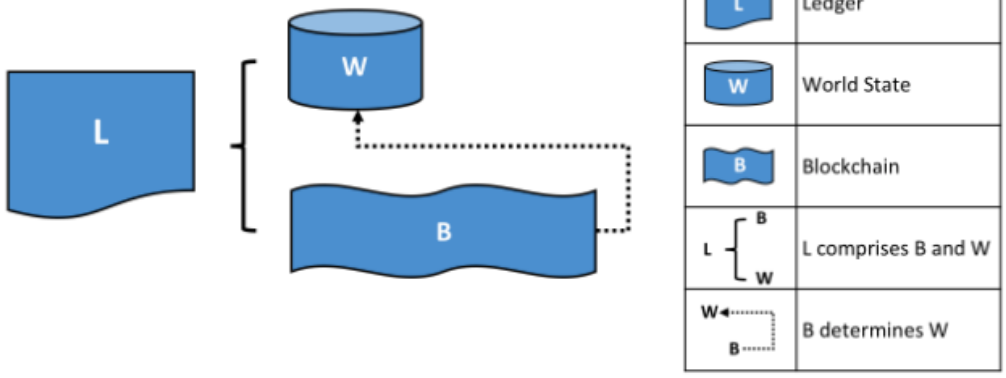
entirely in a future release. Existing users of Solo should move to a single node Raft network for equivalent function.

- **Peer Nodes:**

  Peer nodes maintain a copy of the ledger and execute smart contracts (chaincode). There are two types of peer nodes: *endorsing peers and committing peers.* Endorsing peers simulate and endorse transactions by executing chaincode. They may reside in separate organizations. Committing peers validate transactions, update the ledger, and maintain the blockchain's state. Peers also provide data privacy through the use of private data collections.

- **Ledger:**

  The Ledger is a critical component where all transactions are recorded. It maintains a history of all transactions, making it immutable and tamper-resistant. The ledger is divided into two components: the World State and the Transaction Log (Blockchain).

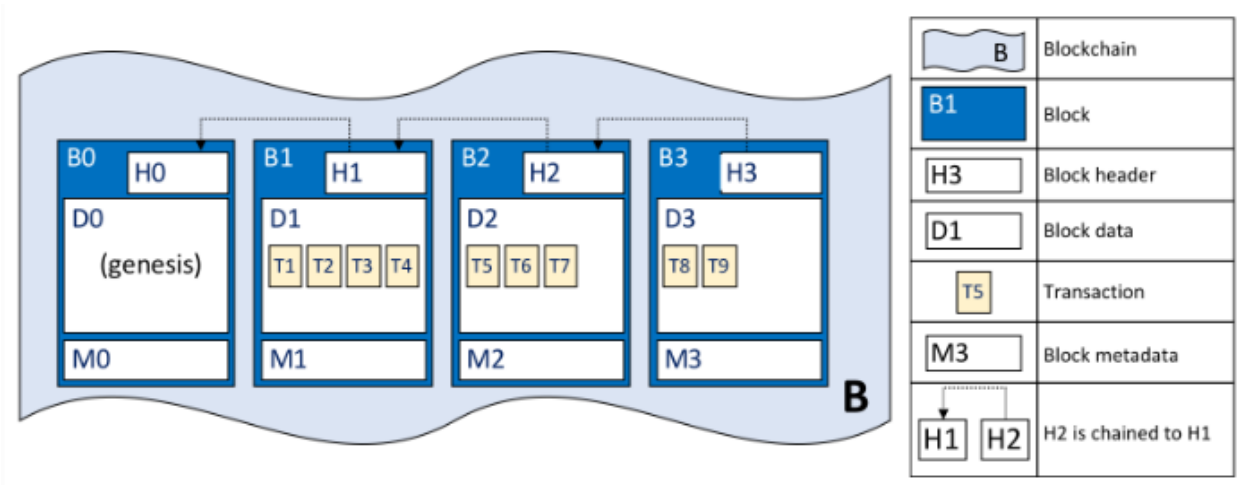


- **Blockchain:**

  The blockchain is structured as a sequential log of interlinked blocks, where each block contains a sequence of transactions, each transaction representing a query or update to the world state.

Each block's header includes a hash of the block's transactions, as well a hash of the prior block's header. In this way, all transactions on the ledger are sequenced and cryptographically linked together. This hashing and linking makes the ledger data very secure. Even if one node hosting the ledger was tampered with, it would not be able to convince all the other nodes that it has the 'correct' blockchain because the ledger is distributed throughout a network of independent nodes.

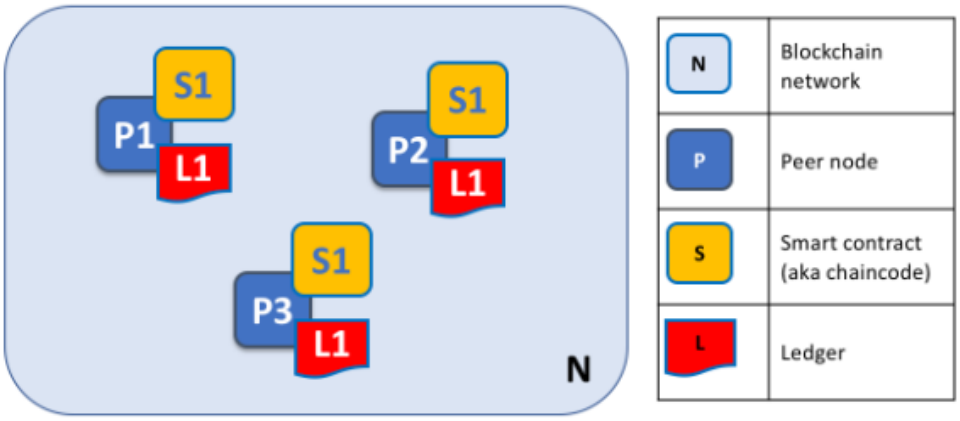
The blockchain is always implemented as a file, in contrast to the world state, which uses a database. This is a sensible design choice as the blockchain data structure is heavily biased towards a very small set of simple operations. Appending to the end of the blockchain is the primary operation, and query is currently a relatively infrequent operation.



A blockchain B containing blocks B0, B1, B2, B3. B0 is the first block in the blockchain, the genesis block.

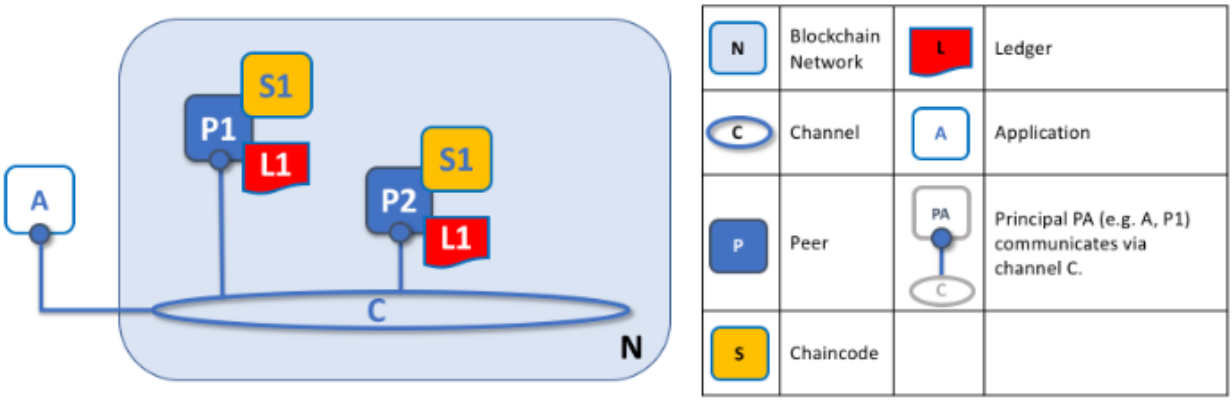
- **Chaincode (Smart Contracts):**

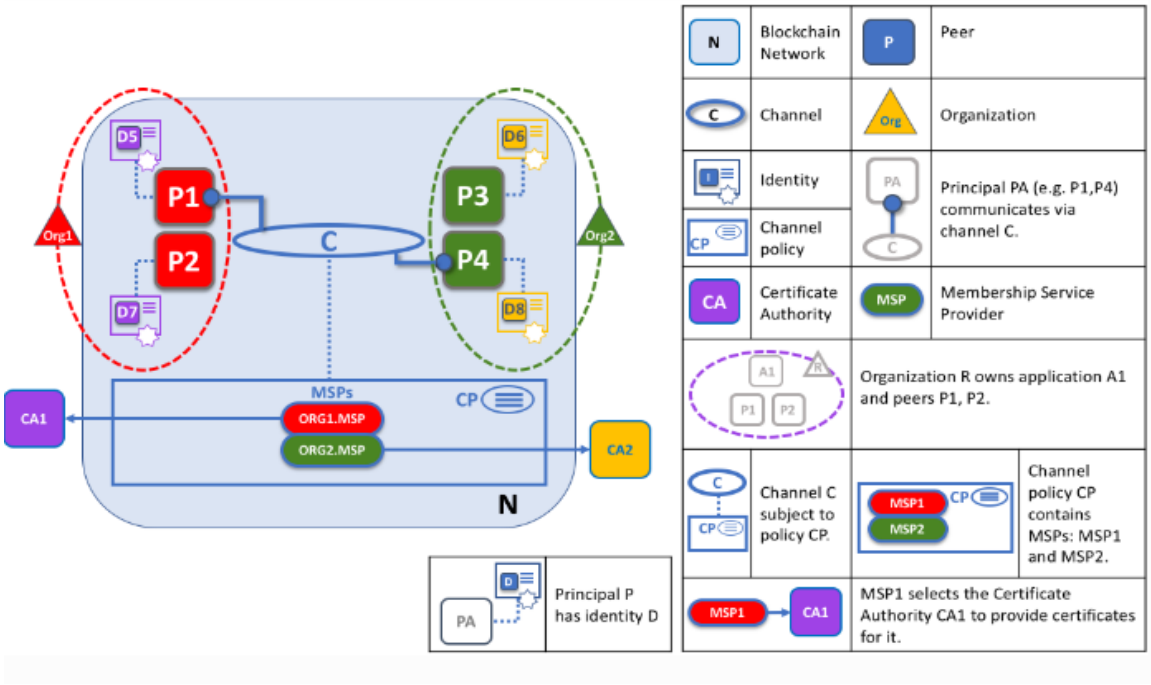
  Chaincode represents smart contracts in Hyperledger Fabric. It defines the business logic and rules for transactions. Chaincode can be written in multiple programming languages, including Go, JavaScript, and Java. Transactions are only valid if they meet the conditions defined in the chaincode.



- **Channels:**

  Channels enable private communication and data sharing between a subset of network participants. Each channel has its own ledger, ensuring data privacy and confidentiality. This feature is particularly useful for scenarios where multiple organizations are part of the same network but should not have access to each other's data.

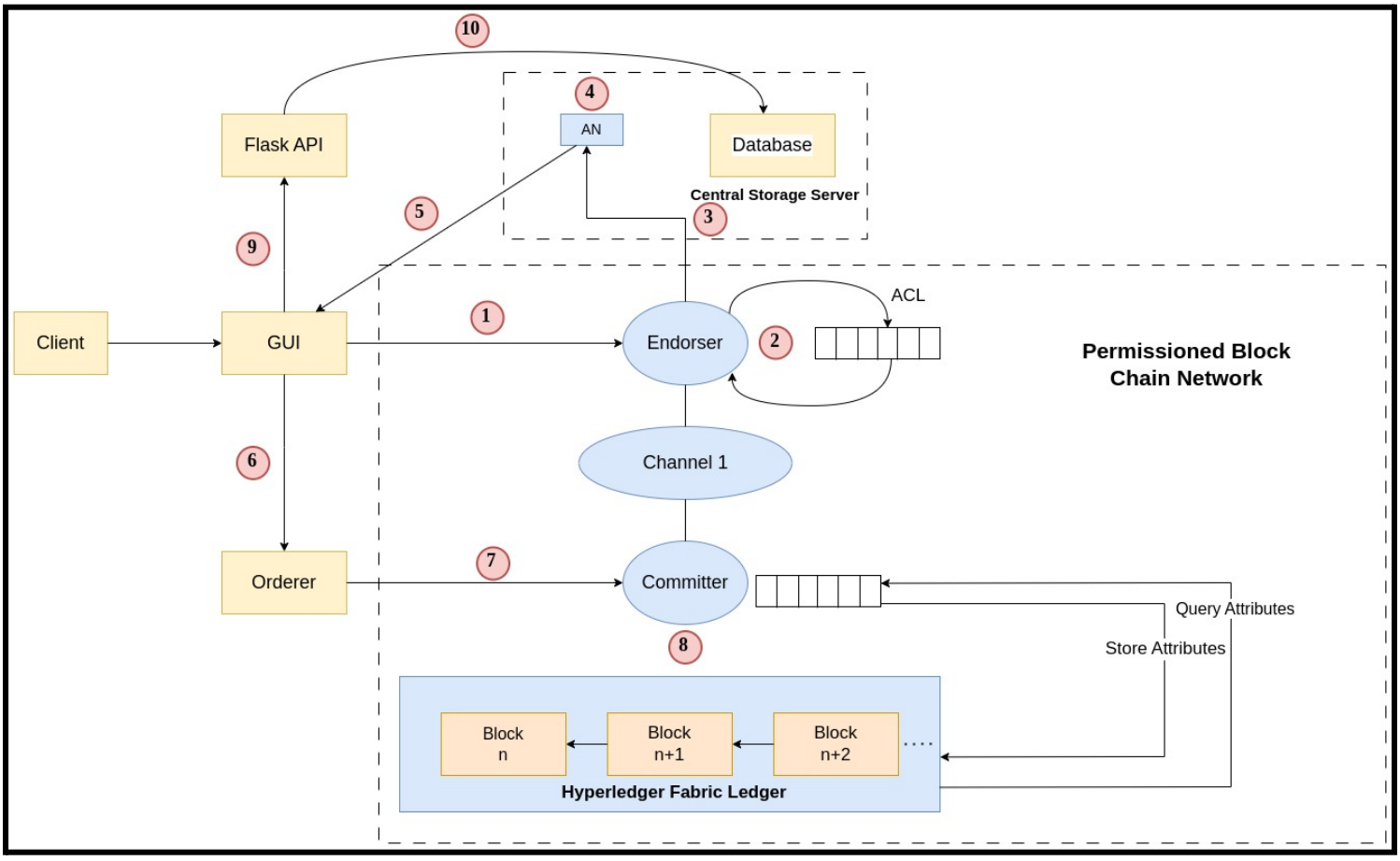



D5
P1
P2
D7
Org1
C
D6
P3
P4
D8
Org2
CA1
MSPs
ORG1.MSP
ORG2.MSP
CP
CA2
N
PA
D
Principal P has identity D
N
Blockchain Network
P
Peer
C
Channel
Org
Organization
I
Identity
PA
C
Principal PA (e.g. P1,P4) communicates via channel C.
CP
Channel policy
CA
Certificate Authority
MSP
Membership Service Provider
A1
R
P1
P2
Organization R owns application A1 and peers P1, P2.
C
CP
Channel C subject to policy CP.
MSP1
CP
MSP2
Channel policy CP contains MSPs: MSP1 and MSP2.
MSP1
CA1
MSP1 selects the Certificate Authority CA1 to provide certificates for it.

CHAPTER - 4
Implementation

Architecture & Design

A top-level architecture, often referred to as a Level 1 architecture, provides a high-level overview of the structure and components of a software system or network. It serves as a foundational blueprint for understanding the system's organization and how its major components interact.



Glossary

1) **GUI:** A GUI, or Graphical User Interface, is a visual interface that allows users to interact with computers or software through graphical elements like icons and buttons. It simplifies user interaction by providing a more intuitive and user-friendly experience compared to text-based interfaces.
2) **Endorser:** An endorser is a trusted entity responsible for validating and endorsing transactions. Endorsers simulate the transaction's execution, sign the results, and submit them to the ordering service. Their consensus ensures the legitimacy of transactions before inclusion in the blockchain.
3) **Committer:** A committer is a node responsible for committing endorsed transactions to the ledger. Committers verify the endorsements, achieve consensus on transaction validity, and then update the shared ledger. They play a crucial role in finalizing and recording transactions on the blockchain.
4) **Orderer:** An orderer is a component that manages the sequence and consistency of transactions. It collects endorsed transactions from endorsers, orders them into a block, and then disseminates the blocks to peers for validation and commitment. Orderers ensure a consistent ledger across all participants.
5) **Ledger:** A ledger is a distributed, tamper-resistant database that maintains a record of all transactions. The ledger is replicated across all nodes (peers) in the network, ensuring transparency and consensus on the state of the system. It includes both the world state, representing the current state of assets, and the transaction history.
6) **Channel:** A channel is a private communication pathway that allows a subset of network members to conduct confidential transactions. It enables participants to create a separate space for secure, private interactions within the larger blockchain network. Channels provide privacy by restricting the visibility of transactions to only the involved parties.
7) **ACL:** An Access Control List (ACL) is a set of rules or permissions associated with an object. An ACL may define who can access certain resources or perform specific actions within the network. It serves as a mechanism to regulate and restrict access to sensitive information, ensuring that only authorized entities have permission to view or modify certain data or execute particular operations. ACLs are crucial for

maintaining security and controlling the permissions granted to users or entities within a network or system.

8) **Flask API:** Flask is a lightweight Python web framework used for building APIs. With a simple syntax, it enables rapid development of RESTful services. Flask provides easy route handling, request processing, and response generation, making it a popular choice for creating scalable and efficient web APIs.
9) **AN:** An authorisation node (AN) is a special node that helps in authorization of the client information and generates a key if the user is valid.

Steps Involved:

1) When the client sends a request to a certain file from the database, the request is sent to the endorser node of the blockchain network via the GUI.
2) The endorser then verifies the validity of the client by executing a chaincode which checks the access control list for the appropriate permissions. The chaincode returns TRUE if the request is valid.
3) Whenever the chaincode execution results in True, a request for key generation is forwarded to the authorization node.
4) The Authorization node is a peer of the blockchain network that is running on the central server. When the request to generate a key is received from the endorser, the AN executes a chaincode that generates a one time use key that gives access to the files stored on the database.
5) The key is forwarded back to the GUI.
6) Parallely, this information is sent to the orderer to order the transaction to be stored on the ledger for future reference.
7) The orderer orders the transaction and sends it to the committer node of the network.
8) The committer node is responsible to submit the transaction to the blockchain and update the ledger. After the update, the updated ledger is replicated on each of the nodes/ peers of the blockchain network.
9) GUI takes the help of Flask API to show the files on the database to the user.

10) The flask API checks for the validity of the key and then connects the client with the database and the user is able to access the files stored on the database using the GUI.

Deployment

Hardware

- **CPU:** A multi-core processor (quad-core or higher) is recommended. Hyperledger Fabric's consensus algorithms and smart contract execution can be CPU-intensive, so having multiple cores can improve performance.
- **RAM:** At least 8 GB of RAM is recommended for running a basic Hyperledger Fabric network. However, more RAM may be necessary for larger networks or complex smart contracts
- **Storage:** You will need sufficient storage space for the following:
  - **Operating System:** Allocate at least 30 GB of storage for the operating system.
  - **Blockchain Data:** Depending on your network's activity, you will need additional storage space to store the blockchain ledger data. This requirement can grow significantly over time, so plan for scalability.
- **Chaincode and Configuration:** Reserve space for storing chaincode packages and network configuration files.
- **Operating System:** Hyperledger Fabric supports Linux-based operating systems, such as Ubuntu or CentOS. Ensure that your chosen OS is compatible with Hyperledger Fabric's version.
- **Network Connectivity:** A stable network connection with sufficient bandwidth is crucial, especially for larger networks with frequent transactions. It's recommended to have a reliable internet connection for communication between nodes.

Software

- **Docker:** Docker is a containerization platform that is essential for running Hyperledger Fabric components in isolated containers. We'll need Docker to create

and manage the containers for the nodes, orderers, and other components of your blockchain network.

- **Docker Compose:** Docker Compose is a tool for defining and running multi-container Docker applications. It is used to define the services, networks, and volumes in a YAML file to set up a multi-container application environment.
- **Go Programming Language:** Hyperledger Fabric is primarily written in Go, and you'll need Go to build and interact with the Fabric source code. A specific version of Go is often recommended, so check the Hyperledger Fabric documentation for the compatible Go version for the release you are using.
- **Node.js (Optional):** If you plan to use Hyperledger Fabric SDKs for Node.js to interact with the blockchain network, you'll need Node.js and npm (Node Package Manager) installed.
- **Git:** Git is a version control system that you'll use to clone and manage the Hyperledger Fabric source code repositories.
- **Python:** Python is used for certain scripting and utility purposes in the Hyperledger Fabric environment. Ensure that you have a compatible version of Python installed.
- **Java (Optional):** If you are using Java-based applications or chaincode, you may need Java Development Kit (JDK) installed.
- **C/C++ Compiler (Optional):** Some components of Hyperledger Fabric may require a C/C++ compiler, especially when building and installing native dependencies.
- **Text Editor or Integrated Development Environment (IDE):** You'll need a text editor or IDE for viewing and editing configuration files, smart contracts (chaincode), and other code-related tasks. For example - VScode
- **Hyperledger Fabric Binaries:** Hyperledger Fabric provides pre-built binary files for various components. You'll need to download and install these binaries as part of setting up your network.

  Please note that the specific software versions and requirements may vary depending on the version of Hyperledger Fabric you intend to use. It's essential to refer to the official *Hyperledger Fabric documentation* and release notes for the most up-to-date and version-specific software requirements and installation instructions.

Installation

Refer to the Hyperledger Fabric implementation documentation [1] developed by me to install various softwares and clone hyperledger repositories.

This is a comprehensive documentation on setting up a test Hyperledger Fabric network on your device. Hyperledger Fabric stands as a robust and versatile blockchain framework, and this guide aims to provide you with a clear and step-by-step process to establish a functional test network. Whether you are a developer, system administrator, or blockchain enthusiast, this documentation is designed to assist you in navigating through the essential steps involved in configuring and deploying a Hyperledger Fabric network locally.

From installing the necessary prerequisites to initializing the network, creating channels, and deploying smart contracts, each section is crafted to provide detailed insights and instructions. By the end of this guide, you will have a fully operational Hyperledger Fabric network running on your device, offering you hands-on experience with this powerful blockchain technology

CHAPTER - 5
Results

Outcome

Starting the Network

The test-network when started does the following things:

- It creates docker containers for each peer and orderer.
- It created identities (Certificates) for each of the nodes of the network.
- You can check the running docker containers by using the command *docker ps -a.*

```
sarthak@vm01:~/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=v2.5.4
DOCKER_IMAGE_VERSION=v2.5.4
/home/sarthak/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
Creating network "fabric_test" with the default driver
Creating volume "compose_orderer.example.com" with default driver
Creating volume "compose_peer0.org1.example.com" with default driver
Creating volume "compose_peer0.org2.example.com" with default driver
Creating orderer.example.com    ... done
Creating peer0.org2.example.com ... done
Creating peer0.org1.example.com ... done
Creating cli                    ... done
```

```
CONTAINER ID   IMAGE                               COMMAND             CREATED         STATUS                  PORTS
                                                                                                NAMES
f9c0061c6e04   hyperledger/fabric-tools:latest     "/bin/bash"         4 seconds ago   Up Less than a second
                                                                                                cli
ffcb5d658e3e   hyperledger/fabric-peer:latest      "peer node start"   9 seconds ago   Up 3 seconds            0.0.0.0:7051->7051/tcp, :::7051
->7051/tcp, 0.0.0.0:9444->9444/tcp, :::9444->9444/tcp                                           peer0.org1.example.com
7f72f4c3f72a   hyperledger/fabric-peer:latest      "peer node start"   9 seconds ago   Up 4 seconds            0.0.0.0:9051->9051/tcp, :::9051
->9051/tcp, 7051/tcp, 0.0.0.0:9445->9445/tcp, :::9445->9445/tcp                                 peer0.org2.example.com
6e910338af56   hyperledger/fabric-orderer:latest   "orderer"           9 seconds ago   Up 3 seconds            0.0.0.0:7050->7050/tcp, :::7050
->7050/tcp, 0.0.0.0:7053->7053/tcp, :::7053->7053/tcp, 0.0.0.0:9443->9443/tcp, :::9443->9443/tcp   orderer.example.com
```

Creating a channel on the Network

After running the command *createChannel*, a channel by the name of mychannel will be created. All the peers or various organizations along with their orderers will be connected to the channel.

```
sarthak@vm01:~/fabric-samples/test-network$ ./network.sh createChannel -c mychannel
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'
Using organization 1
/home/sarthak/fabric-samples/test-network/../bin/configtxgen
+ '[' 0 -eq 1 ']'
+ configtxgen -profile ChannelUsingRaft -outputBlock ./channel-artifacts/mychannel.block -channelID mychannel
2023-11-19 15:32:34.961 IST 0001 INFO [common.tools.configtxgen] main -> Loading configuration
2023-11-19 15:32:34.997 IST 0002 INFO [common.tools.configtxgen.localconfig] completeInitialization -> orderer type: etcdraft
2023-11-19 15:32:34.999 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
2023-11-19 15:32:34.999 IST 0004 INFO [common.tools.configtxgen.localconfig] Load -> Loaded configuration: /home/sarthak/fabric-samples/test-n
etwork/configtx/configtx.yaml
2023-11-19 15:32:35.130 IST 0005 INFO [common.tools.configtxgen] doOutputBlock -> Generating genesis block
2023-11-19 15:32:35.130 IST 0006 INFO [common.tools.configtxgen] doOutputBlock -> Creating application channel genesis block
2023-11-19 15:32:35.158 IST 0007 INFO [common.tools.configtxgen] doOutputBlock -> Writing genesis block
+ res=0
Creating channel mychannel
Adding orderers
+ . scripts/orderer.sh mychannel
+ '[' 0 -eq 1 ']'
+ res=0
Status: 201
{
        "name": "mychannel",
        "url": "/participation/v1/channels/mychannel",
        "consensusRelation": "consenter",
        "status": "active",
        "height": 1
```

```
Joining org1 peer to the channel...
Using organization 1
+ peer channel join -b ./channel-artifacts/mychannel.block
+ res=0
2023-11-19 15:32:45.618 IST 0001 INFO [channelCmd] InitCmdFactory -> Endorser and orderer connections initialized
2023-11-19 15:32:46.837 IST 0002 INFO [channelCmd] executeJoin -> Successfully submitted proposal to join channel
Joining org2 peer to the channel...
Using organization 2
+ peer channel join -b ./channel-artifacts/mychannel.block
+ res=0
2023-11-19 15:32:50.425 IST 0001 INFO [channelCmd] InitCmdFactory -> Endorser and orderer connections initialized
2023-11-19 15:32:50.758 IST 0002 INFO [channelCmd] executeJoin -> Successfully submitted proposal to join channel
Setting anchor peer for org1...
Using organization 1
Fetching channel config for channel mychannel
Using organization 1
Fetching the most recent configuration block for the channel
+ peer channel fetch config config_block.pb -o orderer.example.com:7050 --ordererTLSHostnameOverride orderer.example.com -c mychannel --tls --
cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/organizations/ordererOrganizations/example.com/tlsca/tlsca.example.com-cert.pem
2023-11-19 10:02:54.711 UTC 0001 INFO [channelCmd] InitCmdFactory -> Endorser and orderer connections initialized
2023-11-19 10:02:54.831 UTC 0002 INFO [cli.common] readBlock -> Received block: 0
2023-11-19 10:02:54.837 UTC 0003 INFO [channelCmd] fetch -> Retrieving last config block: 0
2023-11-19 10:02:54.845 UTC 0004 INFO [cli.common] readBlock -> Received block: 0
Decoding config block to JSON and isolating config to Org1MSPconfig.json
+ configtxlator proto_decode --input config_block.pb --type common.Block --output config_block.json
+ jq '.data.data[0].payload.data.config' config_block.json
Generating anchor peer update transaction for Org1 on channel mychannel
+ jq '.channel_group.groups.Application.groups.Org1MSP.values += {"AnchorPeers":{"mod_policy": "Admins","value":{"anchor_peers": [{"host": "pe
er0.org1.example.com","port": 7051}]},"version": "0"}}' Org1MSPconfig.json
+ configtxlator proto_encode --input Org1MSPconfig.json --type common.Config --output original_config.pb
+ configtxlator proto_encode --input Org1MSPmodified_config.json --type common.Config --output modified_config.pb
+ configtxlator compute_update --channel_id mychannel --original original_config.pb --updated modified_config.pb --output config_update.pb
+ configtxlator proto_decode --input config_update.pb --type common.ConfigUpdate --output config_update.json
```

```
Decoding config block to JSON and isolating config to Org2MSPconfig.json
+ configtxlator proto_decode --input config_block.pb --type common.Block --output config_block.json
+ jq '.data.data[0].payload.data.config' config_block.json
Generating anchor peer update transaction for Org2 on channel mychannel
+ jq '.channel_group.groups.Application.groups.Org2MSP.values += {"AnchorPeers":{"mod_policy": "Admins","value":{"anchor_peers": [{"host": "pe
er0.org2.example.com","port": 9051}]},"version": "0"}}' Org2MSPconfig.json
+ configtxlator proto_encode --input Org2MSPconfig.json --type common.Config --output original_config.pb
+ configtxlator proto_encode --input Org2MSPmodified_config.json --type common.Config --output modified_config.pb
+ configtxlator compute_update --channel_id mychannel --original original_config.pb --updated modified_config.pb --output config_update.pb
+ configtxlator proto_decode --input config_update.pb --type common.ConfigUpdate --output config_update.json
+ jq .
++ cat config_update.json
+ echo '{"payload":{"header":{"channel_header":{"channel_id":"mychannel", "type":2}},"data":{"config_update":{' '"channel_id":' '"mychannel",'
 '"isolated_data":' '{},' '"read_set":' '{' '"groups":' '{' '"Application":' '{' '"groups":' '{' '"Org2MSP":' '{' '"groups":' '{},' '"mod_poli
cy":' '"",' '"policies":' '{' '"Admins":' '{' '"mod_policy":' '"",' '"policy":' null, '"version":' '"0"' '},' '"Endorsement":' '{' '"mod_polic
y":' '"",' '"policy":' null, '"version":' '"0"' '},' '"Readers":' '{' '"mod_policy":' '"",' '"policy":' null, '"version":' '"0"' '},' '"Writer
s":' '{' '"mod_policy":' '"",' '"policy":' null, '"version":' '"0"' '}' '},' '"values":' '{' '"MSP":' '{' '"mod_policy":' '"",' '"value":' nul
l, '"version":' '"0"' '}' '},' '"version":' '"0"' '}' '},' '"mod_policy":' '"",' '"policies":' '{},' '"values":' '{},' '"version":' '"0"' '}'
'},' '"mod_policy":' '"",' '"policies":' '{},' '"values":' '{},' '"version":' '"0"' '},' '"write_set":' '{' '"groups":' '{' '"Application":' '
{' '"groups":' '{' '"Org2MSP":' '{' '"groups":' '{},' '"mod_policy":' '"Admins",' '"policies":' '{' '"Admins":' '{' '"mod_policy":' '"",' '"po
licy":' 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"' '}' '},' '"v
ersion":' '"1"' '}' '},' '"mod_policy":' '"",' '"policies":' '{},' '"values":' '{},' '"version":' '"0"' '}' '},' '"mod_policy":' '"",' '"polic
ies":' '{},' '"values":' '{},' '"version":' '"0"' '}' '}}}}'
+ configtxlator proto_encode --input config_update_in_envelope.json --type common.Envelope --output Org2MSPanchors.tx
2023-11-19 10:03:01.059 UTC 0001 INFO [channelCmd] InitCmdFactory -> Endorser and orderer connections initialized
2023-11-19 10:03:01.153 UTC 0002 INFO [channelCmd] update -> Successfully submitted channel update
Anchor peer set for org 'Org2MSP' on channel 'mychannel'
Channel 'mychannel' joined
sarthak@vm01:~/fabric-samples/test-network$
```

Initializing Chaincode

This command packages the chaincode and then installs the chaincode on all the peers of the channel *mychannel*.

```
sarthak@vm01:~/fabric-samples/test-network$
sarthak@vm01:~/fabric-samples/test-network$
sarthak@vm01:~/fabric-samples/test-network$
sarthak@vm01:~/fabric-samples/test-network$ ./network.sh deployCC -ccn basic -ccp ../asset-transfer-basic/chaincode-go -ccl go
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-go
- CC_SRC_LANGUAGE: go
- 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
Vendoring Go dependencies at ../asset-transfer-basic/chaincode-go
~/fabric-samples/asset-transfer-basic/chaincode-go ~/fabric-samples/test-network
~/fabric-samples/test-network
Finished vendoring Go dependencies
+ peer lifecycle chaincode package basic.tar.gz --path ../asset-transfer-basic/chaincode-go --lang golang --label basic_1.0
+ res=0
++ peer lifecycle chaincode calculatepackageid basic.tar.gz
+ PACKAGE_ID=basic_1.0:e4de097efb5be42d96aebc4bde18eea848aad0f5453453ba2aad97f2e41e0d57
Chaincode is packaged
Installing chaincode on peer0.org1...
Using organization 1
+ peer lifecycle chaincode queryinstalled --output json
+ grep '^basic_1.0:e4de097efb5be42d96aebc4bde18eea848aad0f5453453ba2aad97f2e41e0d57$'
+ jq -r 'try (.installed_chaincodes[].package_id)'
```

```
~/fabric-samples/asset-transfer-basic/chaincode-go ~/fabric-samples/test-network
~/fabric-samples/test-network
Finished vendoring Go dependencies
+ peer lifecycle chaincode package basic.tar.gz --path ../asset-transfer-basic/chaincode-go --lang golang --label basic_1.0
+ res=0
++ peer lifecycle chaincode calculatepackageid basic.tar.gz
+ PACKAGE_ID=basic_1.0:e4de097efb5be42d96aebc4bde18eea848aad0f5453453ba2aad97f2e41e0d57
Chaincode is packaged
Installing chaincode on peer0.org1...
Using organization 1
+ peer lifecycle chaincode queryinstalled --output json
+ grep '^basic_1.0:e4de097efb5be42d96aebc4bde18eea848aad0f5453453ba2aad97f2e41e0d57$'
+ jq -r 'try (.installed_chaincodes[].package_id)'
+ test 1 -ne 0
+ peer lifecycle chaincode install basic.tar.gz
+ res=0
2023-11-19 15:37:43.424 IST 0001 INFO [cli.lifecycle.chaincode] submitInstallProposal -> Installed remotely: response:<status:200 payload:"\nJ
basic_1.0:e4de097efb5be42d96aebc4bde18eea848aad0f5453453ba2aad97f2e41e0d57\022\tbasic_1.0" >
2023-11-19 15:37:43.458 IST 0002 INFO [cli.lifecycle.chaincode] submitInstallProposal -> Chaincode code package identifier: basic_1.0:e4de097e
fb5be42d96aebc4bde18eea848aad0f5453453ba2aad97f2e41e0d57
Chaincode is installed on peer0.org1
Install chaincode on peer0.org2...
Using organization 2
+ grep '^basic_1.0:e4de097efb5be42d96aebc4bde18eea848aad0f5453453ba2aad97f2e41e0d57$'
+ peer lifecycle chaincode queryinstalled --output json
+ jq -r 'try (.installed_chaincodes[].package_id)'
+ test 1 -ne 0
+ peer lifecycle chaincode install basic.tar.gz
+ res=0
2023-11-19 15:39:42.316 IST 0001 INFO [cli.lifecycle.chaincode] submitInstallProposal -> Installed remotely: response:<status:200 payload:"\nJ
basic_1.0:e4de097efb5be42d96aebc4bde18eea848aad0f5453453ba2aad97f2e41e0d57\022\tbasic_1.0" >
2023-11-19 15:39:42.345 IST 0002 INFO [cli.lifecycle.chaincode] submitInstallProposal -> Chaincode code package identifier: basic_1.0:e4de097e
fb5be42d96aebc4bde18eea848aad0f5453453ba2aad97f2e41e0d57
```

```
+ peer lifecycle chaincode queryinstalled --output json
+ jq -r 'try (.installed_chaincodes[].package_id)'
+ test 1 -ne 0
+ peer lifecycle chaincode install basic.tar.gz
+ res=0
2023-11-19 15:39:42.316 IST 0001 INFO [cli.lifecycle.chaincode] submitInstallProposal -> Installed remotely: response:<status:200 payload:"\nJ
basic_1.0:e4de097efb5be42d96aebc4bde18eea848aad0f5453453ba2aad97f2e41e0d57\022\tbasic_1.0" >
2023-11-19 15:39:42.345 IST 0002 INFO [cli.lifecycle.chaincode] submitInstallProposal -> Chaincode code package identifier: basic_1.0:e4de097e
fb5be42d96aebc4bde18eea848aad0f5453453ba2aad97f2e41e0d57
Chaincode is installed on peer0.org2
Using organization 1
+ peer lifecycle chaincode queryinstalled --output json
+ jq -r 'try (.installed_chaincodes[].package_id)'
+ grep '^basic_1.0:e4de097efb5be42d96aebc4bde18eea848aad0f5453453ba2aad97f2e41e0d57$'
+ res=0
basic_1.0:e4de097efb5be42d96aebc4bde18eea848aad0f5453453ba2aad97f2e41e0d57
Query installed successful on peer0.org1 on channel
Using organization 1
+ peer lifecycle chaincode approveformyorg -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile /home/sarthak/fab
ric-samples/test-network/organizations/ordererOrganizations/example.com/tlsca/tlsca.example.com-cert.pem --channelID mychannel --name basic --
version 1.0 --package-id basic_1.0:e4de097efb5be42d96aebc4bde18eea848aad0f5453453ba2aad97f2e41e0d57 --sequence 1
+ res=0
2023-11-19 15:39:45.901 IST 0001 INFO [chaincodeCmd] ClientWait -> txid [50875cf4f7e19e4b73cb09bd4218a24b6155078aa032417f95a18f312b7af65b] com
mitted with status (VALID) at localhost:7051
Chaincode definition approved on peer0.org1 on channel 'mychannel'
Using organization 1
Checking the commit readiness of the chaincode definition on peer0.org1 on channel 'mychannel'...
Attempting to check the commit readiness of the chaincode definition on peer0.org1, Retry after 3 seconds.
+ peer lifecycle chaincode checkcommitreadiness --channelID mychannel --name basic --version 1.0 --sequence 1 --output json
+ res=0
{
        "approvals": {
                "Org1MSP": true
```

```
Checking the commit readiness of the chaincode definition successful on peer0.org1 on channel 'mychannel'
Using organization 2
Checking the commit readiness of the chaincode definition on peer0.org2 on channel 'mychannel'...
Attempting to check the commit readiness of the chaincode definition on peer0.org2, Retry after 3 seconds.
+ peer lifecycle chaincode checkcommitreadiness --channelID mychannel --name basic --version 1.0 --sequence 1 --output json
+ res=0
{
        "approvals": {
                "Org1MSP": true,
                "Org2MSP": true
        }
}
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/sarthak/fabric-sampl
es/test-network/organizations/ordererOrganizations/example.com/tlsca/tlsca.example.com-cert.pem --channelID mychannel --name basic --peerAddre
sses localhost:7051 --tlsRootCertFiles /home/sarthak/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com/tlsca/tlsca.
org1.example.com-cert.pem --peerAddresses localhost:9051 --tlsRootCertFiles /home/sarthak/fabric-samples/test-network/organizations/peerOrgani
zations/org2.example.com/tlsca/tlsca.org2.example.com-cert.pem --version 1.0 --sequence 1
+ res=0
2023-11-19 15:40:05.197 IST 0001 INFO [chaincodeCmd] ClientWait -> txid [dc88f7983cd1045f5ecdfe716371cdad23895bb29c587f9307a3a08d2d902ec2] com
mitted with status (VALID) at localhost:9051
2023-11-19 15:40:05.219 IST 0002 INFO [chaincodeCmd] ClientWait -> txid [dc88f7983cd1045f5ecdfe716371cdad23895bb29c587f9307a3a08d2d902ec2] com
mitted 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]
```

```
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/sarthak/fabric-sampl
es/test-network/organizations/ordererOrganizations/example.com/tlsca/tlsca.example.com-cert.pem --channelID mychannel --name basic --peerAddre
sses localhost:7051 --tlsRootCertFiles /home/sarthak/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com/tlsca/tlsca.
org1.example.com-cert.pem --peerAddresses localhost:9051 --tlsRootCertFiles /home/sarthak/fabric-samples/test-network/organizations/peerOrgani
zations/org2.example.com/tlsca/tlsca.org2.example.com-cert.pem --version 1.0 --sequence 1
+ res=0
2023-11-19 15:40:05.197 IST 0001 INFO [chaincodeCmd] ClientWait -> txid [dc88f7983cd1045f5ecdfe716371cdad23895bb29c587f9307a3a08d2d902ec2] com
mitted with status (VALID) at localhost:9051
2023-11-19 15:40:05.219 IST 0002 INFO [chaincodeCmd] ClientWait -> txid [dc88f7983cd1045f5ecdfe716371cdad23895bb29c587f9307a3a08d2d902ec2] com
mitted 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
sarthak@vm01:~/fabric-samples/test-network$
```

Invoking the Chaincode

This command is used to interact with the ledger.

```
sarthak@vm01:~/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
sarthak@vm01:~/fabric-samples/test-network$ peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --c
afile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychan
nel -n basic --peerAddresses localhost:7051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.examp
le.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":[]}'

2023-11-19 15:46:06.267 IST 0001 INFO [chaincodeCmd] chaincodeInvokeOrQuery -> Chaincode invoke successful. result: status:200
sarthak@vm01:~/fabric-samples/test-network$
```

Querying the Chaincode

This command will query the ledger for certain information.

```
arthak@vm01:~/fabric-samples/test-network$ peer chaincode query -C mychannel -n basic -c '{"Args":["GetAllAssets"]}'

[{"AppraisedValue":300,"Color":"blue","ID":"asset1","Owner":"Tomoko","Size":5},{"AppraisedValue":400,"Color":"red","ID":"asset2","Owner":"Brad
","Size":5},{"AppraisedValue":500,"Color":"green","ID":"asset3","Owner":"Jin Soo","Size":10},{"AppraisedValue":600,"Color":"yellow","ID":"asse
t4","Owner":"Max","Size":10},{"AppraisedValue":700,"Color":"black","ID":"asset5","Owner":"Adriana","Size":15},{"AppraisedValue":800,"Color":"w
hite","ID":"asset6","Owner":"Michel","Size":15}]
arthak@vm01:~/fabric-samples/test-network$
```

Adding a peer to the existing network

Creating identities

```
❯ ./organizations/fabric-ca/registerPeer1Org1.sh
2021/11/10 22:52:54 [INFO] Configuration file location: /Users/adityajoshi/fabric-samples/test-network/organizations/peerOrganizations/or
g1.example.com/fabric-ca-client-config.yaml
2021/11/10 22:52:54 [INFO] TLS Enabled
2021/11/10 22:52:54 [INFO] TLS Enabled
Password: peer1pw
2021/11/10 22:52:54 [INFO] TLS Enabled
2021/11/10 22:52:54 [INFO] generating key: &{A:ecdsa S:256}
2021/11/10 22:52:54 [INFO] encoded CSR
2021/11/10 22:52:54 [INFO] Stored client certificate at /Users/adityajoshi/fabric-samples/test-network/organizations/peerOrganizations/or
g1.example.com/peers/peer1.org1.example.com/msp/signcerts/cert.pem
2021/11/10 22:52:54 [INFO] Stored root CA certificate at /Users/adityajoshi/fabric-samples/test-network/organizations/peerOrganizations/o
rg1.example.com/peers/peer1.org1.example.com/msp/cacerts/localhost-7054-ca-org1.pem
2021/11/10 22:52:54 [INFO] Stored Issuer public key at /Users/adityajoshi/fabric-samples/test-network/organizations/peerOrganizations/org
1.example.com/peers/peer1.org1.example.com/msp/IssuerPublicKey
2021/11/10 22:52:54 [INFO] Stored Issuer revocation public key at /Users/adityajoshi/fabric-samples/test-network/organizations/peerOrgani
zations/org1.example.com/peers/peer1.org1.example.com/msp/IssuerRevocationPublicKey
2021/11/10 22:52:54 [INFO] TLS Enabled
2021/11/10 22:52:54 [INFO] generating key: &{A:ecdsa S:256}
2021/11/10 22:52:54 [INFO] encoded CSR
2021/11/10 22:52:55 [INFO] Stored client certificate at /Users/adityajoshi/fabric-samples/test-network/organizations/peerOrganizations/or
g1.example.com/peers/peer1.org1.example.com/tls/signcerts/cert.pem
2021/11/10 22:52:55 [INFO] Stored TLS root CA certificate at /Users/adityajoshi/fabric-samples/test-network/organizations/peerOrganizatio
ns/org1.example.com/peers/peer1.org1.example.com/tls/tlscacerts/tls-localhost-7054-ca-org1.pem
2021/11/10 22:52:55 [INFO] Stored Issuer public key at /Users/adityajoshi/fabric-samples/test-network/organizations/peerOrganizations/org
1.example.com/peers/peer1.org1.example.com/tls/IssuerPublicKey
2021/11/10 22:52:55 [INFO] Stored Issuer revocation public key at /Users/adityajoshi/fabric-samples/test-network/organizations/peerOrgani
zations/org1.example.com/peers/peer1.org1.example.com/tls/IssuerRevocationPublicKey
```

Creating docker container

```
❯ docker-compose -f docker/docker-compose-peer1org1.yaml up -d
WARNING: Found orphan containers (couchdb0, orderer.example.com, couchdb1, cli, peer0.org1.example.com, ca_orderer, ca_org1, ca_org2, pee
r0.org2.example.com) for this project. If you removed or renamed this service in your compose file, you can run this command with the --r
emove-orphans flag to clean it up.
Creating peer1.org1.example.com ... done
```

Challenges

During the course of my internship at Space Application Centre (SAC), Indian Space Research Organisation (ISRO), I encountered several challenges that pushed me to learn, adapt, and grow. These challenges, while sometimes daunting, provided valuable learning experiences:

1) **Understanding Hyperledger Fabric**: One of the significant challenges I faced was comprehending the intricacies of Hyperledger Fabric, a complex and evolving blockchain technology framework. The decentralized nature of blockchain and its unique architecture posed a steep learning curve. However, with determination and guidance, I was able to make progress and gain a deeper understanding of this technology.
2) **Limited Resources and Guidance:** Being in a relatively new and emerging field, I encountered a scarcity of resources and guidance materials specific to my work. The lack of readily available references made research and problem-solving more demanding. This challenge motivated me to be resourceful and creative in finding solutions and seeking guidance from colleagues and mentors whenever possible.
3) **Limited Knowledge of Networking:** My background in networking was limited, and this posed a challenge since my work often involved network-related tasks. Understanding how networks functioned, including the protocols and architectures, was essential to my internship responsibilities. I had to invest extra time and effort in self-study to bridge this knowledge gap.
4) **Hardware-Related Issues:** In a technology-intensive environment like SAC, hardware problems occasionally disrupted my workflow. This included issues with computer systems, servers, or other equipment. These interruptions, while frustrating, forced me to develop troubleshooting skills and the ability to adapt to unforeseen technical challenges.

Despite these challenges, my internship at SAC was a valuable and enriching experience. Overcoming these obstacles not only enhanced my technical skills but also fostered resilience and problem-solving abilities. These challenges have contributed significantly to

my personal and professional growth, and I am grateful for the opportunity to have faced and conquered them during my internship.

Sincerely,

Sarthak Aggarwal

CHAPTER - 6
Conclusion

Potential Use Cases

Hyperledger Fabric and private blockchain technology, when integrated with other technologies, offer innovative solutions across various domains. In supply chain management, the combination of Hyperledger Fabric and IoT devices ensures transparency and traceability. Healthcare data management benefits from AI and ML integration, enabling secure sharing and analysis of patient data. Energy trading in microgrids becomes efficient with blockchain and network communication, enabling secure peer-to-peer transactions. Intellectual property protection is enhanced by integrating Hyperledger Fabric with digital rights management, ensuring secure registration and enforcement of rights. Tamper-proof voting systems are achieved by combining blockchain with ultraviolet communication. Cross-border trade finance is streamlined using blockchain, AI, and IoT, automating processes and enhancing risk assessment. Food safety is improved through blockchain, RFID, and ML, ensuring traceability and early contamination detection. Real estate transactions are simplified with blockchain's transparency and smart contracts, while educational credentials verification benefits from blockchain and AI integration. Legal contracts are revolutionized by smart contracts, automating execution and improving accuracy through AI. These use cases showcase the versatility of Hyperledger Fabric, enhancing security, transparency, and efficiency in various industries.

Challenges

During my internship at Space Application Centre (SAC), Indian Space Research Organisation (ISRO), I encountered various challenges that greatly contributed to my learning and growth. These challenges included mastering the complexities of Hyperledger Fabric, a challenging blockchain technology framework, navigating limited resources and guidance in an emerging field, addressing my limited knowledge of networking, and dealing

with hardware-related disruptions. Despite the difficulties, overcoming these obstacles not only enhanced my technical skills but also fostered resilience and problem-solving abilities, leading to a valuable and enriching internship experience.

Future Plans

After the conclusion of my internship at the Space Application Centre (SAC), Indian Space Research Organisation (ISRO), my future plans revolve around advancing the ongoing project to align with the specific requirements at ISRO. Building upon the foundation laid during my internship, my focus will be on delving deeper into the intricacies of Hyperledger Fabric network implementation. This entails acquiring an in-depth understanding of the technology, exploring its nuances, and mastering the intricacies of deploying and managing a secure and efficient blockchain network.

A significant aspect of my future work involves the development of a system dedicated to sensitive data access control. Leveraging the capabilities of a private blockchain network using the binaries of Hyperledger Fabric, I aim to create a robust framework that ensures the confidentiality and integrity of sensitive information. This system will not only cater to the stringent data security requirements at ISRO but also contribute to the broader landscape of secure data management. The overarching goal is to integrate the principles of blockchain technology, particularly Hyperledger Fabric, to establish a secure and tamper-proof platform for managing sensitive data, aligning with the high standards of technological innovation upheld by ISRO.

References

1) **NFT and Access control**

   https://chat.openai.com/share/2dcfb793-90a6-4d79-a01d-8bbe82debf27

2) **Hyperledger Fabric Documentation**

   https://hyperledger-fabric.readthedocs.io/en/release-2.5/

3) **GitHub Repository**

   https://github.com/hyperledger/fabric/blob/release-2.5/docs/source/index.rst

Annexure

1) **Step-to-step procedure for HLF Implementation**

   Hyperledger Fabric Implementation

2) **Understanding Ethereum Blockchain network**

   Ethereum Test Network Documentation

3) **Understanding HLF CLI commands**

   Detailed overview of Implementation process

Annexure 1

Hyperledger Fabric Test Network Implementation Documentation

INDEX

INTRODUCTION

This is a comprehensive documentation on setting up a test Hyperledger Fabric network on your device. Hyperledger Fabric stands as a robust and versatile blockchain framework, and this guide aims to provide you with a clear and step-by-step process to establish a functional test network. Whether you are a developer, system administrator, or blockchain enthusiast, this documentation is designed to assist you in navigating through the essential steps involved in configuring and deploying a Hyperledger Fabric network locally.

From installing the necessary prerequisites to initializing the network, creating channels, and deploying smart contracts, each section is crafted to provide detailed insights and instructions. By the end of this guide, you will have a fully operational Hyperledger Fabric network running on your device, offering you hands-on experience with this powerful blockchain technology.

1. References

- https://hyperledger-fabric.readthedocs.io/en/latest/test_network.html
- https://github.com/srthkaggrwl/HLF_Implementation.git

Update & Upgrade Ubuntu

```
sudo apt update
sudo apt ugrade
```

Installing git, curl & docker-compose

```
sudo apt-get install git curl docker-compose -y
```

Start & Enable Docker

```
sudo systemctl start docker
sudo usermod -a -G docker sarthak
docker --version
docker-compose --version
sudo systemctl enable docker
```

Note: The test network has been successfully verified with Docker Desktop **version 2.5.0.1** and is the recommended version at this time. Higher versions may not work.

- ## Verify the installation

```
curl --version
git —version
```

- ## Installing jq:
- Install the latest version of jq

```
sudo apt-get install -y jq
```

Verify the installation and check the version of jq using the below command.

```
jq --version
```

- ## Installing python

```
sudo apt-get install python3
```

Verify the installation and check the version of Git using the below command.

```
python3 --version
```

- Install Go
- Install the latest version of go
- # https://go.dev/doc/install

```
cd ~
wget https://go.dev/dl/go1.21.1.linux-amd64.tar.gz
sudo rm -rf /usr/local/go && tar -C ~ -xzf go1.21.1.linux-amd64.tar.gz
```

Add ~/go/bin to the PATH environment variable.
You can do this by adding the following line to your $HOME**/.profile** or /etc/profile (for a system-wide installation):

```
export PATH
=$PATH:~/go/bin
```

Note: Changes made to a profile file may not apply until the next time you log into your computer. To apply the changes immediately, just run the shell commands directly or execute them from the profile using a command such as source $HOME/.profile.

Verify the installation and check the version of Git using the below command.

```
go version
```

Hyperledger fabric Installation

- Configure Git*

if you do decide to install Git on Windows and manage the Fabric repositories natively (as opposed to within WSL2 and its Git installation), then make sure you configure Git as follows:

```
git config --global core.autocrlf false
git config --global core.longpaths true
```

- Clone the repository.

```
curl -sSLO
https://raw.githubusercontent.com/hyperledger/fabric/main/scripts/instal
```

```
l-fabric.sh && chmod +x install-fabric.sh

./install-fabric.sh -h

./install-fabric.sh docker samples binary
```

NOTE:

1) You may encounter the below issue when you run the above command.

```
failed to get default registry endpoint from daemon (Got permission
denied while trying to connect to the Docker daemon socket at
unix:///var/run/docker.sock:
```

To fix this you need to run the command below.

```
sudo chmod 666 /var/run/docker.sock
```

NOTE: 1) You may receive the following error while installing the fabrics

```
Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is
the docker daemon running?
```

To fix the error, install the latest version of Docker. Then clear the docker container and install again.

```
docker rm -f $(docker ps -aq)
docker rmi -f $(docker images -q)
docker network prune
```

Hyperledger fabric Test Network Implementation

- Navigate to the test-network directory

```
cd fabric-samples/test-network
```

- Starting your Test-Network

```
./network.sh -h                    // to print the script help text
./network.sh down
./network.sh up
```

If any error occurs regarding that the docker images are already present, Clear any existing docker images

```
./network.sh down
docker rm -f $(docker ps -aq)
docker rmi -f $(docker images -q)
docker network prune
```

This will re-set up the docker container and delete all the pre-existing images or networks in the container.
After you do this, try starting your network again.

For any more errors refer to Troubleshooting

- Check Docker Containers

```
docker ps -a
```

```
output:
            peer0.org2.example.com
            peer0.org1.example.com
            orderer.example.com
```

Note:
Add ~/fabric-samples/bin to the PATH environment variable.
You also need to set the FABRIC_CFG_PATH to point to the core.yaml file in the fabric-samples repository.
You can do this by adding the following lines to your $HOME/.profile or /etc/profile (for a system-wide installation):

```
export PATH=$PATH:~/fabric-samples/bin
export FABRIC_CFG_PATH=~/fabric-samples/config/
```

Note: Changes made to a profile file may not apply until the next time you log into your computer. To apply the changes immediately, just run the shell commands directly or execute them from the profile using a command such as source $HOME/.profile.

- Channels
- Creating new channels

```
./network.sh createChannel
./network.sh createChannel -c channel1
./network.sh createChannel -c channel2
```

Chain-Code Deployment

- Deploy the chaincode package containing the smart contract by calling the ./network.sh script with the chaincode name and language options.

```
./network.sh deployCC -ccn basic -ccp ../asset-transfer-basic/chaincode-
go -ccl go
```

This script uses the chaincode lifecycle to package, install, query installed chaincode, approve chaincode for both Org1 and Org2, and finally commit the chaincode.

Interacting with the network

```
PWD=~/sarthak/fabric-samples/test-network
```

Organization 1

Setting up environment variables (global variables)

Execute on console.
Set the environment variables that allow you to operate the peer CLI as Org1:

```
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.exam
ple.com/users/Admin@org1.example.com/msp
export CORE_PEER_ADDRESS=localhost:7051
```

Initialize the ledger with assets

Run the following command to initialize the ledger with assets. (Note the CLI does not access the Fabric Gateway peer, so each endorsing peer must be specified.)
:

```
peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride
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
"${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org
1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --
tlsRootCertFiles
"${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org
2.example.com/tls/ca.crt" -c '{"function":"InitLedger","Args":[]}'
```

```
output:
2023-09-19 05:15:04.640 UTC 0001 INFO [chaincodeCmd]
chaincodeInvokeOrQuery -> Chaincode invoke successful. result:
status:200
```

List the Assets

You can now query the ledger from your CLI. Run the following command to get the list of assets that were added to your channel ledger:

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

```
output (formatted using jsonformatter.com):

[
                {
                    "AppraisedValue": 300,
                    "Color": "blue",
                    "ID": "asset1",
                    "Owner": "Tomoko",
                    "Size": 5
                },
                {
                    "AppraisedValue": 400,
                    "Color": "red",
                    "ID": "asset2",
                    "Owner": "Brad",
                    "Size": 5
                },
                {
                    "AppraisedValue": 500,
                    "Color": "green",
                    "ID": "asset3",
                    "Owner": "Jin Soo",
                    "Size": 10
                },
                {
                    "AppraisedValue": 600,
```

```
                "Color": "yellow",
                "ID": "asset4",
                "Owner": "Max",
                "Size": 10
            },
            {
                "AppraisedValue": 700,
                "Color": "black",
                "ID": "asset5",
                "Owner": "Adriana",
                "Size": 15
            },
            {
                "AppraisedValue": 800,
                "Color": "white",
                "ID": "asset6",
                "Owner": "Michel",
                "Size": 15
            }
        ]
```

Invoking the Chain-codes

Chain Codes are invoked when a network member wants to transfer or change an asset on the ledger. Use the following command to change the owner of an asset on the ledger by invoking the asset-transfer (basic) chaincode:

```
peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride
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.org
1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --
tlsRootCertFiles
"${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org
2.example.com/tls/ca.crt" -c
'{"function":"TransferAsset","Args":["asset6","Christopher"]}'
```

This command changes the owner of an asset on the ledger by invoking the asset-transfer (basic) chaincode

```
output:

2023-09-19 05:24:01.738 UTC 0001 INFO [chaincodeCmd]
chaincodeInvokeOrQuery -> Chaincode invoke successful. result:
status:200 payload:"Michel"
```

As the change should be visible to each node of each organization, let’s check the ledger of node1 of organization 2.

Organization 2

Setting up environment variables (global variables)

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

As the ledger is already initialized on the network, let’s just query the ledger for asset info.

Querying the Ledger

```
peer chaincode query -C mychannel -n basic -c
'{"Args":["ReadAsset","asset6"]}'
```

```
output:

{"AppraisedValue":800,"Color":"white","ID":"asset6","Owner":"Christopher
","Size":15}
```

We can see that the Owner of asset6 changed from Michel to Christopher

Bringing Down the Network

```
./network.sh down
```

Adding Org3 to the existing network

Starting Network

Starting the network and creating certificates for the nodes, then creating a channel named channel1

```
 ./network.sh up createChannel -c channel1
```

Generate the Org3 Crypto Material

```
cd addOrg3
../../bin/cryptogen generate --config=org3-crypto.yaml --
output="../organizations"
```

Deploying Chain code

```
./network.sh deployCC -ccn basic -ccp ../asset-transfer-basic/chaincode-
go -ccl go

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

export
ORDERER_CA=${PWD}/organizations/ordererOrganizations/example.com/orderer
s/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem

export
CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org1.exam
ple.com/users/Admin@org1.example.com/msp

export CORE_PEER_ADDRESS=localhost:7051

export
CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/peerOrganizations/org1.
example.com/peers/peer0.org1.example.com/tls/ca.crt

export CORE_PEER_LOCALMSPID=Org1MSP
```

Invoking and Query Chain Code

Invoke

```
peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride
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.org
1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --
tlsRootCertFiles
"${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org
2.example.com/tls/ca.crt" -c
'{"Args":["CreateAsset","501","red","20","aditya","100"]}'
```

Query

```
peer chaincode query -C mychannel -n basic -c
'{"Args":["ReadAsset","501"]}'
```

Org3 Setup

AddOrg3 directory

```
cd addOrg3/
```

Generate ca certificates for Org3

```
./addOrg3.sh generate -ca
```

Generate org config

```
export FABRIC_CFG_PATH=$PWD
../../bin/configtxgen -printOrg Org3MSP >
../organizations/peerOrganizations/org3.example.com/org3.json
```

It converts the config.yaml to a json file which is stored in the directory mentioned.

Starting docker container

```
docker-compose -f compose/docker/docker-compose-org3.yaml up -d
```

setup ENV for Org3

```
cd ..

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

Fetch config

```
peer channel fetch config channel-artifacts/config_block.pb -o
localhost:7050 --ordererTLSHostnameOverride orderer.example.com -c
mychannel --tls --cafile
${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.e
xample.com/msp/tlscacerts/tlsca.example.com-cert.pem
```

We fetch the config block and storing is as an output in channel-artifacts/config_block.pb.
We also specify the channel name, orderer, and tls certs.

Decoding config block and trimming it

```
configtxlator proto_decode --input ./channel-artifacts/config_block.pb -
-type common.Block --output ./channel-artifacts/config_block.json
jq ".data.data[0].payload.data.config" ./channel-
artifacts/config_block.json > ./channel-artifacts/config.json
```

We decode the config_block.pb file into json format and store it in the location channel-artifacts/config_block.json
This file contains the configuration for the whole block

A file name config.json is also created containing just the config part of the config_block.json Now we will add the Org3.json file to the config file so that we have the configuration of Org3 in config_block.json

Adding config

```
jq -s '.[0] * {"channel_group":{"groups":{"Application":{"groups":
{"Org3MSP":.[1]}}}}}' ./channel-artifacts/config.json
./organizations/peerOrganizations/org3.example.com/org3.json >
./channel-artifacts/modified_config.json
```

This command creates a modified config.json file named modified_config.json in the channel-artifacts directory which contains the configuration of Org3 as well.

Translate config.json back into a protobuf called config.pb

```
configtxlator proto_encode --input ./channel-artifacts/config.json --
type common.Config --output ./channel-artifacts/config.pb
```

We encode the config.json file to protobuf format to calculate the delta.

Encode modified_config.json to modified_config.pb

```
configtxlator proto_encode --input ./channel-
artifacts/modified config.json --type common.Config --output ./channel-
artifacts/modified_config.pb
```

We encode the modified_config.json file to protobuf format to calculate the delta.

Calculate the delta between these two config protobufs

In the context of the configtxlator compute_update command in Hyperledger Fabric, the term "delta" refers to the difference or change between two channel configurations. It represents the set of modifications made to the original configuration to transform it into the updated configuration.

```
configtxlator compute_update --channel_id mychannel --original
./channel-artifacts/config.pb --updated ./channel-
artifacts/modified_config.pb --output ./channel-artifacts/org3_update.pb
```

Org3_update.pb is created which contains the difference in configuration of two files namely – original file i.e config.pb and the updated file modified_config.pb

Decode this object into editable JSON format and call it org3_update.json

```
configtxlator proto_decode --input ./channel-artifacts/org3_update.pb --
type common.ConfigUpdate --output ./channel-artifacts/org3_update.json
```

Converting org3_update.pb to org3_update.json

Adding headers back

Now as we initially trimmed some part of the config-block file to add the org3, we would have to add the headers back to the org3.json file to submit the transaction.

```
echo
'{"payload":{"header":{"channel_header":{"channel_id":"'mychannel'",
"type":2}},"data":{"config_update":'$(cat ./channel-
artifacts/org3_update.json)'}}}' | jq . > ./channel-
artifacts/org3_update_in_envelope.json
```

convert it into the fully fledged protobuf format

```
configtxlator proto_encode --input ./channel-
artifacts/org3_update_in_envelope.json --type common.Envelope --output
./channel-artifacts/org3_update_in_envelope.pb
```

Sign the tx

Sign from Org1

```
peer channel signconfigtx -f channel-
artifacts/org3_update_in_envelope.pb
```

Signing the transaction by org1, peer0

Env for org2

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

```
export CORE_PEER_ADDRESS=localhost:9051
```

Sign from Org2

```
peer channel update -f channel-artifacts/org3_update_in_envelope.pb -c
mychannel -o localhost:7050 --ordererTLSHostnameOverride
orderer.example.com --tls --cafile
${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.e
xample.com/msp/tlscacerts/tlsca.example.com-cert.pem
```

Signing and updating the channel with the latest channel configuration

Join Org3 to the Channel

Env

```
export CORE_PEER_TLS_ENABLED=true
export CORE_PEER_LOCALMSPID="Org3MSP"
export
CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/peerOrganizations/org3.
example.com/peers/peer0.org3.example.com/tls/ca.crt
export
CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org3.exam
ple.com/users/Admin@org3.example.com/msp
export CORE_PEER_ADDRESS=localhost:11051
```

Fetch block 0

```
peer channel fetch 0 channel-artifacts/mychannel.block -o localhost:7050
--ordererTLSHostnameOverride orderer.example.com -c mychannel --tls --
cafile
${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.e
xample.com/msp/tlscacerts/tlsca.example.com-cert.pem
```

Join channel

```
peer channel list //will give the list of channels connected to thispeer

peer channel join -b channel-artifacts/mychannel.block

peer channel getinfo -c mychannel
```

Chaincode setup

Install CC

```
peer lifecycle chaincode install basic.tar.gz
peer lifecycle chaincode queryinstalled
```

Approve CC

```
export CC_PACKAGE_ID=

peer lifecycle chaincode approveformyorg -o localhost:7050 --
ordererTLSHostnameOverride orderer.example.com --tls --cafile
${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.e
xample.com/msp/tlscacerts/tlsca.example.com-cert.pem --channelID
mychannel --name basic --version 1.0 --package-id $CC_PACKAGE_ID --
sequence 1
```

Query Commited

```
peer lifecycle chaincode querycommitted --channelID mychannel --name
basic --cafile
${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.e
xample.com/msp/tlscacerts/tlsca.example.com-cert.pem
```

Invoke CC

```
peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride
orderer.example.com --tls --cafile
${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.e
xample.com/msp/tlscacerts/tlsca.example.com-cert.pem -C mychannel -n
basic --peerAddresses localhost:9051 --tlsRootCertFiles
${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2
.example.com/tls/ca.crt --peerAddresses localhost:11051 --
tlsRootCertFiles
${PWD}/organizations/peerOrganizations/org3.example.com/peers/peer0.org3
.example.com/tls/ca.crt -c '{"function":"InitLedger","Args":[]}'
```

Query CC

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

Adding a peer to the existing network

Add the necessary files

Add the Registerpeer1.sh file (Appendix 1.1) in the following directory.

```
./test-network/organizations/fabric-ca/
```

Add the docker-compose-peer1org1.sh file (Appendix 1.2) in the following directory.

```
./test-network/compose/docker/
```

Setting up the new peer

Creating MSP Identities for peer1.org1.example.com

```
./organizations/fabric-ca/registerPeer1.sh
```

Starting up the peer container

```
docker-compose -f compose/docker/docker-compose-peer1.yaml up -d
```

Joining existing channel

Query channel on peer0.org1.example.com

```
peer channel list
peer channel fetch -c mychannel newest
CORE_PEER_ADDRESS=localhost:8051 peer channel getinfo -c mychannel
```

Query channel on peer1.org1.example.com

```
CORE_PEER_ADDRESS=localhost:8051 peer channel list
CORE_PEER_ADDRESS=localhost:8051 peer channel fetch -c mychannel newest
CORE_PEER_ADDRESS=localhost:8051 peer channel getinfo -c mychannel
```

Join channel

```
CORE_PEER_ADDRESS=localhost:8051 peer channel join -b ./channel-
artifacts/mychannel.block
```

Query channel on peer1.org1.example.com

```
CORE_PEER_ADDRESS=localhost:8051 peer channel list
CORE_PEER_ADDRESS=localhost:8051 peer channel fetch -c mychannel newest
CORE_PEER_ADDRESS=localhost:8051 peer channel getinfo -c mychannel
```

Chaincode Setup

Install CC

```
export CC_NAME=basic
CORE_PEER_ADDRESS=localhost:8051 peer lifecycle chaincode install
${CC_NAME}.tar.gz
```

Query installed CC

```
CORE_PEER_ADDRESS=localhost:8051 peer lifecycle chaincode queryinstalled
```

Invoke CC

```
CORE_PEER_ADDRESS=localhost:8051 peer chaincode invoke -n basic -C
mychannel -o localhost:7050 --ordererTLSHostnameOverride
orderer.example.com  --tls --cafile "$ORDERER_CA" --peerAddresses
localhost:8051 --tlsRootCertFiles
${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer1.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 '{"Args":["CreateAsset", "200","red",
"20","aditya","100"]}'
```

Query CC

```
CORE_PEER_ADDRESS=localhost:8051 peer chaincode query -n basic -C
mychannel -o localhost:7050 --ordererTLSHostnameOverride
orderer.example.com  -c '{"Args":["ReadAsset", "200"]}'
```

Appendices

1.1 Registerpeer1.sh

```
export PATH=$PATH:${PWD}/../bin
export
FABRIC_CA_CLIENT_HOME=${PWD}/organizations/peerOrganizations/org1.exampl
e.com/
fabric-ca-client register --caname ca-org1 --id.name peer1 --id.secret
peer1pw --id.type peer --tls.certfiles ${PWD}/organizations/fabric-
ca/org1/tls-cert.pem
mkdir -p
organizations/peerOrganizations/org1.example.com/peers/peer1.org1.exampl
e.com
fabric-ca-client enroll -u https://peer1:peer1pw@localhost:7054 --caname
ca-org1 -M
${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer1.org1
.example.com/msp --csr.hosts peer1.org1.example.com --tls.certfiles
${PWD}/organizations/fabric-ca/org1/tls-cert.pem
cp
${PWD}/organizations/peerOrganizations/org1.example.com/msp/config.yaml
${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer1.org1
.example.com/msp/config.yaml
fabric-ca-client enroll -u https://peer1:peer1pw@localhost:7054 --caname
ca-org1 -M
${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer1.org1
.example.com/tls --enrollment.profile tls --csr.hosts
peer1.org1.example.com --csr.hosts localhost --tls.certfiles
${PWD}/organizations/fabric-ca/org1/tls-cert.pem
cp
${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer1.org1
.example.com/tls/tlscacerts/*
${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer1.org1
.example.com/tls/ca.crt
cp
${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer1.org1
.example.com/tls/signcerts/*
${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer1.org1
.example.com/tls/server.crt
cp
${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer1.org1
.example.com/tls/keystore/*
${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer1.org1
.example.com/tls/server.key
```

1.2 docker-compose-peer1org1.yaml

```
# Copyright IBM Corp. All Rights Reserved.
#
# SPDX-License-Identifier: Apache-2.0
#

version: "3.7"

volumes:
  peer1.org1.example.com:

networks:
  test:
    name: fabric_test

services:
  peer1.org1.example.com:
    container_name: peer1.org1.example.com
    image: hyperledger/fabric-peer:latest
    labels:
      service: hyperledger-fabric
    environment:
      #Generic peer variables
      - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
      - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=fabric_test
      - FABRIC_LOGGING_SPEC=INFO
      #- FABRIC_LOGGING_SPEC=DEBUG
      - CORE_PEER_TLS_ENABLED=true
      - CORE_PEER_PROFILE_ENABLED=true
      - CORE_PEER_TLS_CERT_FILE=/etc/hyperledger/fabric/tls/server.crt
      - CORE_PEER_TLS_KEY_FILE=/etc/hyperledger/fabric/tls/server.key
      - CORE_PEER_TLS_ROOTCERT_FILE=/etc/hyperledger/fabric/tls/ca.crt
      # Peer specific variabes
      - CORE_PEER_ID=peer1.org1.example.com
      - CORE_PEER_ADDRESS=peer1.org1.example.com:8051
      - CORE_PEER_LISTENADDRESS=0.0.0.0:8051
      - CORE_PEER_CHAINCODEADDRESS=peer1.org1.example.com:8052
      - CORE_PEER_CHAINCODELISTENADDRESS=0.0.0.0:8052
      - CORE_PEER_GOSSIP_BOOTSTRAP=peer1.org1.example.com:8051
      - CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer1.org1.example.com:8051
      - CORE_PEER_LOCALMSPID=Org1MSP
      - CORE_OPERATIONS_LISTENADDRESS=0.0.0.0:18051
    volumes:
      - /var/run/docker.sock:/host/var/run/docker.sock
      -
../organizations/peerOrganizations/org1.example.com/peers/peer1.org1.exa
mple.com/msp:/etc/hyperledger/fabric/msp
      -
../organizations/peerOrganizations/org1.example.com/peers/peer1.org1.exa
mple.com/tls:/etc/hyperledger/fabric/tls
      - peer1.org1.example.com:/var/hyperledger/production
```

```
    working_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer
    command: peer node start
    ports:
      - 8051:8051
      - 18051:18051
    networks:
      - test
```

Annexure 2

Understanding Ethereum Test Network Documentation

- Installing Metamask
  - Visit Metamask and download the metamask wallet and add the extension to your Google account.
  - Sign up to create an account.
    Note: Store your 12-word character key safely to retrieve your account in future. Don't share it with anyone as anyone with this can get full access to all your accounts on metamask.

- Connecting to a Test-Network
  - Check the Github Repository for the recommended test network to be used.
  - At the writing of this document, the recommended test network was - Sepolia
  - Visit Sepolia Etherscan to see the details of your metamask account.
  - Now check the github repository for the main test faucet to get your test ether
  - Sign in and connect the faucet to your metamask account to get your test Sepolia ether.
  - You can check your transaction on etherscan using the public hash of your account.

- Blockchain Demo
  - Visit Demo to understand what a blockchain is and how it works.
  - See the implementation of various concepts like keys, signatures and transactions.

- References
  - https://www.youtube.com/watch?v=gyMwXuJrbJQ
  - https://www.youtube.com/watch?v=MGemhK9t44Q

Annexure 3

Detailed overview of Implementation process

./network.sh up

Generating Certificates using cryptogen tool

You are using the `cryptogen` tool to generate cryptographic materials (certificates and keys) for the organizations and entities within your Hyperledger Fabric network.

Creating Org1 Identities

You are generating cryptographic identities for the first organization (`Org1`) specified in the configuration file `crypto-config-org1.yaml`.
The result (`res=0`) indicates that this operation was successful.

Creating Org2 Identities

You are generating cryptographic identities for the second organization (`Org2`) specified in the configuration file `crypto-config-org2.yaml`.
The result (`res=0`) indicates that this operation was successful.

Creating Orderer Org Identities

You are generating cryptographic identities for the orderer organization specified in the configuration file `crypto-config-orderer.yaml`.
The result (`res=0`) indicates that this operation was successful.

Generating CCP Files for Org1 and Org2

CCP (Connection Profile) files are used to configure client applications to interact with Hyperledger Fabric networks. You are generating these files for both `Org1` and `Org2`.

Creating Docker Containers

You are using Docker Compose to create Docker containers for various components of your Hyperledger Fabric network, including the orderer, peers for `Org1` and `Org2`, and a CLI container.
The containers are named `orderer.example.com`, `peer0.org1.example.com`, `peer0.org2.example.com`, and `cli`.

Creating Volumes

Docker volumes are being created to store persistent data for the Docker containers. These volumes are named `compose_orderer.example.com`, `compose_peer0.org1.example.com`, and `compose_peer0.org2.example.com`.

Creating network "fabric_test" with the default driver

You are creating a Docker network named "fabric_test" to facilitate communication between the containers in your Fabric network.

Container Creation Status

"done" indicates that the Docker containers have been successfully created for the respective components of your Hyperledger Fabric network.

./network.sh createChannel

Channel Creation

The process starts with the creation of a channel named 'mychannel' using the `peer channel create` command. This step generates a channel configuration transaction.

Genesis Block Generation

After the channel is created, a channel genesis block ('mychannel.block') is generated using the `configtxgen` tool. This block contains the initial configuration for the channel.


```
{
    "name": "mychannel",
    "url": "/participation/v1/channels/mychannel",
    "consensusRelation": "consenter",
    "status": "active",
    "height": 1
```

}

Adding Orderers

The orderers for the channel are added using the `scripts/orderer.sh` script. This step configures the orderer nodes to be part of the 'mychannel' and submits the configuration update to the orderer.

Adding orderers
+ . scripts/orderer.sh mychannel
+ '[' 0 -eq 1 ']'
+ res=0

Joining Peers to the Channel

The peers from 'org1' and 'org2' are joined to the channel 'mychannel' using the `peer channel join` command. This step allows these peers to participate in the channel's transactions.

org1
+ peer channel join -b ./channel-artifacts/mychannel.block
+ res=0
2023-09-20 16:45:30.559 IST 0001 INFO [channelCmd] InitCmdFactory -> Endorser and orderer connections initialized
2023-09-20 16:45:30.576 IST 0002 INFO [channelCmd] executeJoin -> Successfully submitted proposal to join channel

Org1
Joining org2 peer to the channel...
Using organization 2
+ peer channel join -b ./channel-artifacts/mychannel.block
+ res=0
2023-09-20 16:45:33.640 IST 0001 INFO [channelCmd] InitCmdFactory -> Endorser and orderer connections initialized
2023-09-20 16:45:33.655 IST 0002 INFO [channelCmd] executeJoin -> Successfully submitted proposal to join channel

Setting Anchor Peers

Anchor peers are set for 'org1' and 'org2' organizations on 'mychannel.' Anchor peers are used for cross-organization communication. The anchor peer configuration is updated using the `configtxlator` tool and submitted to the orderer.

Setting anchor peer for org1...

Channel Configuration Update

The channel configuration is updated to include the anchor peer information for both organizations ('org1' and 'org2'). This update ensures that all peers in the channel are aware of the anchor peers.

```
Using organization 1
Fetching channel config for channel mychannel
Using organization 1
Fetching the most recent configuration block for the channel
+ peer channel fetch config config_block.pb -o orderer.example.com:7050 --
ordererTLSHostnameOverride orderer.example.com -c mychannel --tls --cafile
/opt/gopath/src/github.com/hyperledger/fabric/peer/organizations/ordererOrganizations/example
.com/tlsca/tlsca.example.com-cert.pem
2023-09-20 11:15:33.839 UTC 0001 INFO [channelCmd] InitCmdFactory -> Endorser and
orderer connections initialized
2023-09-20 11:15:33.842 UTC 0002 INFO [cli.common] readBlock -> Received block: 0
2023-09-20 11:15:33.842 UTC 0003 INFO [channelCmd] fetch -> Retrieving last config block: 0
2023-09-20 11:15:33.843 UTC 0004 INFO [cli.common] readBlock -> Received block: 0
+ configtxlator proto_decode --input config_block.pb --type common.Block --output
config_block.json
Decoding config block to JSON and isolating config to Org1MSPconfig.json
+ jq '.data.data[0].payload.data.config' config_block.json
Generating anchor peer update transaction for Org1 on channel mychannel
+ jq '.channel_group.groups.Application.groups.Org1MSP.values +=
{"AnchorPeers":{"mod_policy": "Admins","value":{"anchor_peers": [{"host":
"peer0.org1.example.com","port": 7051}]},"version": "0"}}' Org1MSPconfig.json
+ configtxlator proto_encode --input Org1MSPconfig.json --type common.Config --output
original_config.pb
+ configtxlator proto_encode --input Org1MSPmodified_config.json --type common.Config --
output modified_config.pb
+ configtxlator compute_update --channel_id mychannel --original original_config.pb --updated
modified_config.pb --output config_update.pb
+ configtxlator proto_decode --input config_update.pb --type common.ConfigUpdate --output
config_update.json
+ jq .
```

Finalization and Confirmation

The channel creation and configuration update processes are finalized, and the channel 'mychannel' is considered active.

2023-09-20 11:15:34.608 UTC 0001 INFO [channelCmd] InitCmdFactory -> Endorser and orderer connections initialized
2023-09-20 11:15:34.622 UTC 0002 INFO [channelCmd] update -> Successfully submitted channel update

Join Confirmation

The log also shows that peers from 'org1' and 'org2' successfully submitted join channel proposals, indicating that they have successfully joined the channel.

Anchor peer set for org 'Org2MSP' on channel 'mychannel'
Channel 'mychannel' joined

./network.sh deployCC -ccn basic -ccp ../asset-transfer-basic/chaincode-go -ccl go

Vendoring Go Dependencies

It starts by vendoring the Go dependencies required for the chaincode. It downloads various Go packages necessary for the chaincode's compilation and execution.

Vendoring Go dependencies at ../asset-transfer-basic/chaincode-go
~/fabric-samples/asset-transfer-basic/chaincode-go ~/fabric-samples/test-network
go: downloading github.com/hyperledger/fabric-contract-api-go v1.2.1
go: downloading github.com/hyperledger/fabric-chaincode-go v0.0.0-20230228194215-b84622ba6a7a
go: downloading github.com/hyperledger/fabric-protos-go v0.3.0
go: downloading github.com/stretchr/testify v1.8.2
go: downloading google.golang.org/protobuf v1.28.1
go: downloading github.com/golang/protobuf v1.5.2
go: downloading google.golang.org/grpc v1.53.0
go: downloading github.com/xeipuuv/gojsonschema v1.2.0
go: downloading github.com/go-openapi/spec v0.20.8
go: downloading github.com/gobuffalo/packr v1.30.1
go: downloading github.com/davecgh/go-spew v1.1.1
go: downloading github.com/pmezard/go-difflib v1.0.0
go: downloading gopkg.in/yaml.v3 v3.0.1
go: downloading golang.org/x/net v0.7.0

```
go: downloading google.golang.org/genproto v0.0.0-20230110181048-
76db0878b65f
go: downloading github.com/xeipuuv/gojsonreference v0.0.0-20180127040603-
bd5ef7bd5415
go: downloading github.com/go-openapi/jsonpointer v0.19.5
go: downloading github.com/go-openapi/jsonreference v0.20.0
go: downloading github.com/go-openapi/swag v0.21.1
go: downloading github.com/gobuffalo/envy v1.10.1
go: downloading github.com/gobuffalo/packd v1.0.1
go: downloading golang.org/x/sys v0.5.0
go: downloading github.com/xeipuuv/gojsonpointer v0.0.0-20190905194746-
02993c407bfb
go: downloading github.com/mailru/easyjson v0.7.7
go: downloading gopkg.in/yaml.v2 v2.4.0
go: downloading github.com/joho/godotenv v1.4.0
go: downloading github.com/rogpeppe/go-internal v1.8.1
go: downloading github.com/josharian/intern v1.0.0
go: downloading golang.org/x/text v0.7.0
~/fabric-samples/test-network
Finished vendoring Go dependencies
```

Packaging the Chaincode

After vendoring, the chaincode is packaged using the `peer lifecycle chaincode package` command. The chaincode package is saved as 'basic.tar.gz,' and its package ID is calculated.

```
+ peer lifecycle chaincode package basic.tar.gz --path ../asset-transfer-basic/chaincode-go --
lang golang --label basic_1.0
+ res=0
++ peer lifecycle chaincode calculatepackageid basic.tar.gz
+
PACKAGE_ID=basic_1.0:e4de097efb5be42d96aebc4bde18eea848aad0f5453453ba2aad97f2e
41e0d57
Chaincode is packaged
```

Installing Chaincode on Peer0.org1 and Peer0.org2

The chaincode package is installed on both 'peer0.org1' and 'peer0.org2' of their respective organizations.

```
Installing chaincode on peer0.org1...
```

```
		Using organization 1
		+ grep
'^basic_1.0:e4de097efb5be42d96aebc4bde18eea848aad0f5453453ba2aad97f2e41e0d57$'
		+ jq -r 'try (.installed_chaincodes[].package_id)'
		+ peer lifecycle chaincode queryinstalled --output json
		+ test 1 -ne 0
		+ peer lifecycle chaincode install basic.tar.gz

		+ res=0
		2023-09-19 05:01:02.059 UTC 0001 INFO [cli.lifecycle.chaincode]
submitInstallProposal -> Installed remotely: response:<status:200
payload:"\nJbasic_1.0:e4de097efb5be42d96aebc4bde18eea848aad0f5453453ba2aad97f2e41e
0d57\022\tbasic_1.0" >
		2023-09-19 05:01:02.065 UTC 0002 INFO [cli.lifecycle.chaincode]
submitInstallProposal -> Chaincode code package identifier:
basic_1.0:e4de097efb5be42d96aebc4bde18eea848aad0f5453453ba2aad97f2e41e0d57
		Chaincode is installed on peer0.org1
		Install chaincode on peer0.org2...
		Using organization 2
		+ grep
'^basic_1.0:e4de097efb5be42d96aebc4bde18eea848aad0f5453453ba2aad97f2e41e0d57$'
		+ jq -r 'try (.installed_chaincodes[].package_id)'
		+ peer lifecycle chaincode queryinstalled --output json
		+ test 1 -ne 0
		+ peer lifecycle chaincode install basic.tar.gz

		+ res=0
		2023-09-19 05:02:09.593 UTC 0001 INFO [cli.lifecycle.chaincode]
submitInstallProposal -> Installed remotely: response:<status:200
payload:"\nJbasic_1.0:e4de097efb5be42d96aebc4bde18eea848aad0f5453453ba2aad97f2e41e
0d57\022\tbasic_1.0" >
		2023-09-19 05:02:09.598 UTC 0002 INFO [cli.lifecycle.chaincode]
submitInstallProposal -> Chaincode code package identifier:
basic_1.0:e4de097efb5be42d96aebc4bde18eea848aad0f5453453ba2aad97f2e41e0d57
		Chaincode is installed on peer0.org2
```

Approving Chaincode Definition

Chaincode definition approval is requested from both organizations (Org1MSP and Org2MSP) using the `peer lifecycle chaincode approveformyorg` command. The command sends the approval request to the orderer for ordering. The transaction is committed with status (VALID).

Using organization 1

```
+ grep
'^basic_1.0:e4de097efb5be42d96aebc4bde18eea848aad0f5453453ba2aad97f2e41e0d57$'
+ jq -r 'try (.installed_chaincodes[].package_id)'
+ peer lifecycle chaincode queryinstalled --output json
+ res=0

basic_1.0:e4de097efb5be42d96aebc4bde18eea848aad0f5453453ba2aad97f2e41e0d57
Query installed successful on peer0.org1 on channel
Using organization 1
+ peer lifecycle chaincode approveformyorg -o localhost:7050 --
ordererTLSHostnameOverride orderer.example.com --tls --cafile /home/sarthak/fabric-
samples/test-
network/organizations/ordererOrganizations/example.com/tlsca/tlsca.example.com-cert.pem --
channelID mychannel --name basic --version 1.0 --package-id
basic_1.0:e4de097efb5be42d96aebc4bde18eea848aad0f5453453ba2aad97f2e41e0d57 --
sequence 1
+ res=0
2023-09-19 05:02:11.942 UTC 0001 INFO [chaincodeCmd] ClientWait -> txid
[e8f2b1771fe34716a0129e79e99cfb11d2d3ad3acbfcb735fc69c443e9bbacd2] committed with
status (VALID) at localhost:7051
Chaincode definition approved on peer0.org1 on channel 'mychannel'
Using organization 1
Checking the commit readiness of the chaincode definition on peer0.org1 on
channel 'mychannel'...
Attempting to check the commit readiness of the chaincode definition on
peer0.org1, Retry after 3 seconds.
+ peer lifecycle chaincode checkcommitreadiness --channelID mychannel --name
basic --version 1.0 --sequence 1 --output json
+ res=0
{
        "approvals": {
                "Org1MSP": true,
                "Org2MSP": false
        }
}
Checking the commit readiness of the chaincode definition successful on
peer0.org1 on channel 'mychannel'
Using organization 2
Checking the commit readiness of the chaincode definition on peer0.org2 on
channel 'mychannel'...
Attempting to check the commit readiness of the chaincode definition on
peer0.org2, Retry after 3 seconds.
+ peer lifecycle chaincode checkcommitreadiness --channelID mychannel --name
basic --version 1.0 --sequence 1 --output json
```

```
+ res=0
{
        "approvals": {
                "Org1MSP": true,
                "Org2MSP": false
        }
}
Checking the commit readiness of the chaincode definition successful on
peer0.org2 on channel 'mychannel'
Using organization 2
+ peer lifecycle chaincode approveformyorg -o localhost:7050 --
ordererTLSHostnameOverride orderer.example.com --tls --cafile /home/sarthak/fabric-
samples/test-
network/organizations/ordererOrganizations/example.com/tlsca/tlsca.example.com-cert.pem --
channelID mychannel --name basic --version 1.0 --package-id
basic_1.0:e4de097efb5be42d96aebc4bde18eea848aad0f5453453ba2aad97f2e41e0d57 --
sequence 1
+ res=0
2023-09-19 05:02:20.255 UTC 0001 INFO [chaincodeCmd] ClientWait -> txid
[f7fb21c68518b68fb89c76f7e45cf84075209d17fd50e35156ec9e54a929cc55] committed with
status (VALID) at localhost:9051
Chaincode definition approved on peer0.org2 on channel 'mychannel'
Using organization 1
Checking the commit readiness of the chaincode definition on peer0.org1 on
channel 'mychannel'...
Attempting to check the commit readiness of the chaincode definition on
peer0.org1, Retry after 3 seconds.
+ peer lifecycle chaincode checkcommitreadiness --channelID mychannel --name
basic --version 1.0 --sequence 1 --output json
+ res=0
{
        "approvals": {
                "Org1MSP": true,
                "Org2MSP": true
        }
}
Checking the commit readiness of the chaincode definition successful on
peer0.org1 on channel 'mychannel'
Using organization 2
Checking the commit readiness of the chaincode definition on peer0.org2 on
channel 'mychannel'...
Attempting to check the commit readiness of the chaincode definition on
peer0.org2, Retry after 3 seconds.
```

```
+ peer lifecycle chaincode checkcommitreadiness --channelID mychannel --name basic --version 1.0 --sequence 1 --output json
+ res=0
{
        "approvals": {
                "Org1MSP": true,
                "Org2MSP": true
        }
}
Checking the commit readiness of the chaincode definition successful on peer0.org2 on channel 'mychannel'
```

Checking Commit Readiness

Commit readiness is checked for both organizations using the `peer lifecycle chaincode checkcommitreadiness` command. The output indicates whether each organization has approved the chaincode definition.

```
Using organization 1
Checking the commit readiness of the chaincode definition on peer0.org1 on channel 'mychannel'...
Attempting to check the commit readiness of the chaincode definition on peer0.org1, Retry after 3 seconds.
+ peer lifecycle chaincode checkcommitreadiness --channelID mychannel --name basic --version 1.0 --sequence 1 --output json
+ res=0
{
        "approvals": {
                "Org1MSP": true,
                "Org2MSP": false
        }
}
Checking the commit readiness of the chaincode definition successful on peer0.org1 on channel 'mychannel'
Using organization 2
Checking the commit readiness of the chaincode definition on peer0.org2 on channel 'mychannel'...
Attempting to check the commit readiness of the chaincode definition on peer0.org2, Retry after 3 seconds.
+ peer lifecycle chaincode checkcommitreadiness --channelID mychannel --name basic --version 1.0 --sequence 1 --output json
```

```
+ res=0
{
        "approvals": {
                "Org1MSP": true,
                "Org2MSP": false
        }
}
```

Checking the commit readiness of the chaincode definition successful on peer0.org2 on channel 'mychannel'

Committing Chaincode Definition

The chaincode definition is committed to the channel using the `peer lifecycle chaincode commit` command. This command sends the commit request to the orderer for ordering. The transaction is committed with status (VALID) on both 'peer0.org1' and 'peer0.org2.'

```
Using organization 1
Using organization 2
+ peer lifecycle chaincode commit -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile /home/sarthak/fabric-samples/test-network/organizations/ordererOrganizations/example.com/tlsca/tlsca.example.com-cert.pem --channelID mychannel --name basic --peerAddresses localhost:7051 --tlsRootCertFiles /home/sarthak/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com/tlsca/tlsca.org1.example.com-cert.pem --peerAddresses localhost:9051 --tlsRootCertFiles /home/sarthak/fabric-samples/test-network/organizations/peerOrganizations/org2.example.com/tlsca/tlsca.org2.example.com-cert.pem --version 1.0 --sequence 1
+ res=0
2023-09-19 05:02:28.868 UTC 0001 INFO [chaincodeCmd] ClientWait -> txid [641451943ac5720d95961a9effe298dc738122bf70b6cc345e5023fee875459a] committed with status (VALID) at localhost:9051
2023-09-19 05:02:28.870 UTC 0002 INFO [chaincodeCmd] ClientWait -> txid [641451943ac5720d95961a9effe298dc738122bf70b6cc345e5023fee875459a] committed with status (VALID) at localhost:7051
Chaincode definition committed on channel 'mychannel'
```

Querying Chaincode Definition

Chaincode definition is queried on both 'peer0.org1' and 'peer0.org2' to verify that it has been committed successfully. The output shows the chaincode version, sequence, endorsement plugin, validation plugin, and approvals from both organizations.

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

The final message states that chaincode initialization is not required, indicating that the chaincode has been successfully deployed and committed to the channel.

Chaincode initialization is not required