Introduction to HyperLedger

# History of Hyperledger

The hyperledger project was created in 2015 by the Linux Foundation. It’s goal is to promote the use of the Blockchain in various aspects of business, such as supply chain management, management of intangible assets, such as mortgage deeds and other legal documents and other potential applications.

The aim is to introduce increased reliability and transparency across these systems so that they work more effectively across cross-border transactions. However, one application that was specifically stated that the foundation will not build is a cryptocurrency. I.e., there will never be a *hypercoin* or a other such currency as part of the Hyperledger project.

Currently there are twenty members of the governing board as well as a large number of technology , financial services and other companies contributing technology and infrastructure to the project.

There are a number of projects inside the Hyperledger umbrella.

* Hyperledger Fabric. Hyperledger Fabric is a permissioned blockchain, originally developed by IBM. This infrastructure provides a modular architecture with a strict definition of the different nodes on the network. Additionally, it introduces Smart Contracts called *Chaincode* as well as configurable consensus and membership services.
* Hyperledger Sawtooth. Sawtooth is a blockchain infrastructure created by Intel corporation. Sawtooth employs a consensus mechanism called *Proof of Elapsed Time*.
* Hyperledger Iroha. An infrastructure based on Fabric with a specific focus towards mobile applications.
* Hyperledger Composer. Composer is a rapid prototyping tool with a web based GUI that runs on top of Fabric. Composer is used to quickly set up and prototype applications running on the Fabric blockchain infrastructure.
* Hyperledger Explorer. A blockchain analytics toolset created by IBM, Intel and DTCC.

# The Hyperledger Fabric

The Fabric is the contribution made by IBM to the Hyperledger project. The goal of this project is to enable a modular, open and flexible approach towards building blockchain networks. These features can then allow for for achieving scalability, privacy and other attributes as necessary. The blockchain transactions in the fabric are private, confidential and anonymous for the general user, but can still be traced and linked to users by authorized individuals (auditors). Note that the Fabric can be implemented as a *permissioned* network. With a permissioned network, only authorized users are allowed to transact across this network,.

# The Fabric Architecture

The Fabric is logically organized into three main categories based on the type of service provided. These include membership services, blockchain services, and chaincode services. In the following section, all these categories and associated components are discussed in detail.

## Membership services

These services are used to provide access control capability for the users of the fabric network. The following list shows the functions that membership services perform:

## User identity validation.

### User registration.

Assign appropriate permissions to the users depending on their roles.

### Membership services

Membership services makes use of Public Key Infrastructure (PKI) in order to support identity management and authorization operations. Membership services are made up of various components:

Registration authority (RA)

A service that authenticates the users and assesses the identity of the fabric participants for issuance of certificates.

Enrollment certificate authority

Enrollment certificates (Ecerts) are long term certificates issued by ECA to registered participants in order to provide identification to the entities participating on the network.

Transaction certificate authority

In order to send transactions on the networks, participants are required to hold a transaction certificate. TCA is responsible for issuing transaction certificates to holders of Enrolment certificates and is derived from Ecerts.

TLS certificate authority

In order to secure the network level communication between nodes on the Fabric, TLS certificates are used. TLS certificate authority issues TLS certificates in order to ensure security of the messages being passed between various systems on the blockchain network.

## Blockchain services

Blockchain services are at the core of the Hyperledger Fabric. Components within this category are as follows.

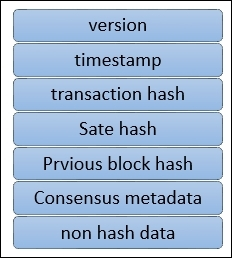
### The Consensus manager

The consensus manager is responsible for providing the interface to the consensus algorithm. This serves as an adapter that receives the transaction from other Hyperledger entities and executes them under criteria according to the type of algorithm chosen. Consensus is pluggable and currently there are three types of consensus algorithm available in Fabric, namely the batch PBFT protocol, SIEVE algorithm, and NOOPS.

### The Distributed ledger

The blockchain and world state are two main elements of the distributed ledger. The blockchain is simply a linked list of blocks (as introduced in earlier chapters) and world ledger is a key-value database. This database is used by smart contracts to store relevant states during execution by the transactions. The blockchain consists of blocks that contain transactions. These transactions contain chaincode, which runs transactions that can result in updating the world state. Each node saves the world state on disk in RocksDB.

Following is an example of a fabric architecture.



Block structure

The fields shown in the preceding diagram are as follows:

Version: Used for keeping track of changes in the protocol.

Timestamp: Timestamp in UTC epoch time, updated by block proposer.

Transaction hash: This field contains the Merkle root hash of the transactions in the block.

State hash: This is the Merkle root hash of the world state.

Previous hash: This is the previous block's hash, which is calculated after serializing the block message and then creating the message digest by applying the SHA3 SHAKE256 algorithm.

Consensus metadata: This is an optional field that can be used by the consensus protocol to provide some relevant information about the consensus.

Non-Hash data: This is some metadata that is stored with the block but is not hashed. This feature makes it possible to have different data on different peers. It also provides the ability to discard data without any impact on the blockchain.

## Ledger storage

In order to save the state of the ledger, RocksDB is used, and it is stored at each peer. RocksDB is a high performance database available at <http://rocksdb.org/>.

## Chaincode services

These services allow the creation of secure containers that are used to execute the chaincode. Components in this category are as follows:

Secure container: Chaincode is deployed in Docker containers that provide a locked down sandboxed environment for smart contract execution. Currently Golang is supported as the main smart contract language, but any other main stream language can be added and enabled if required.

Secure registry: This provides a record of all images containing smart contracts.

### Events

Events on the blockchain can be triggered by validator nodes and smart contracts. External applications can listen to these events and react to them if required via event adapters. They are similar to the concept of events introduced in solidity in the last chapter.

## APIs and CLIs

An application programming interface provides an interface into the fabric by exposing various REST APIs. Additionally, command line interfaces that provide a subset of REST APIs and allow for quick testing and limited interaction with the blockchain are also available.

## Components of the Fabric

There are various components that can be part of the blockchain. These components include but are not limited to the ledger, chaincode, consensus mechanism, access control, events, system monitoring and management, wallets and system integration components.

### Peers or nodes

There are two main types of peers that can be run on a fabric network: Validating and non-validating. Simply put, a validating node runs consensus, creates and validates a transaction, and contributes towards updating the ledger and maintaining the chaincode.

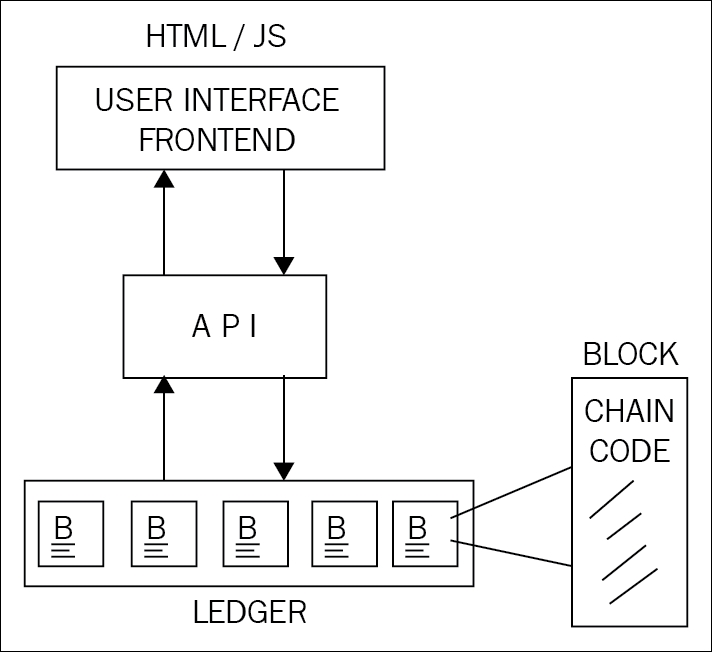
A non-validating peer does not execute transactions and only constructs transactions that are then forwarded to validating nodes.

Both nodes manage and maintain user certificates that have been issued by membership services.

## Applications on blockchain

A typical application on Fabric is simply composed of a user interface, usually written in JavaScript/HTML, that interacts with the backend chaincode (smart contract) stored on the ledger via an API layer.

Following is a diagram showing an example of a fabric application architecture



Hyperledger provides various APIs and command line interfaces to enable interaction with the ledger. These APIs include interfaces for identity, transactions, chaincode, ledger, network, storage, and events.

## Chaincode implementation

Chaincode is usually written in Golang or Java. Chaincode can be public, confidential or access controlled. These codes serve as a smart contract that users can interact with via APIs. Users can call functions in the chaincode that result in a state change, and consequently updates the ledger. There are also functions that are only used to query the ledger and do not result in any state change.

Chaincode implementation is performed by first creating the chaincode shim interface in the code. It can either be in Java or Golang code. The following four functions are required in order to implement the chaincode:

Init(): This function is invoked when chaincode is deployed onto the ledger. This initializes the chaincode and results in making a state change, which accordingly updates the ledger.

Invoke(): This function is used when contracts are executed. It takes a function name as parameters along with an array of arguments. This function results in a state change and writes to the ledger.

Query(): This function is used to query the current state of a deployed chaincode. This function does not make any changes to the ledger.

Main(): This function is executed when a peer deploys its own copy of the chaincode. The chaincode is registered with the peer using this function.

## Application model

Any blockchain application for Hyperledger Fabric follows MVC-B architecture. This is based on the popular MVC design pattern. Components in this model are Model, View, Control, and Blockchain:

View logic: This is concerned with the user interface. It can be a desktop, web application or mobile frontend.

Control logic: This is the orchestrator between user interface, data model, and APIs.

Data model: This model is used to manage the off-chain data.

Blockchain logic: This is used to manage the blockchain via the controller and the data model via transactions.

## Lab 1. Installing a Fabric Network.

The Lab 1 lab notes will detail instructions for installing our first Hyperledger Fabric network.

The supplied shell script byfn.sh has the following components:

1. The Crypto Generator
2. The Configuration Transaction Generator

### The Crypto Generator

Cryptogen consumes a file called crypto-config.yaml - that contains the network topology and allows us to generate a set of certificates and keys for both the Organizations and the components that belong to those Organizations. Each Organization is provisioned a unique root certificate (ca-cert) that binds specific components (peers and orderers) to that Org. By assigning each Organization a unique CA certificate, we are mimicking a typical network where a participating Member would use its own Certificate Authority. Transactions and communications within Hyperledger Fabric are signed by an entity’s private key (keystore), and then verified by means of a public key (signcerts).

You will notice a count variable within this file. We use this to specify the number of peers per Organization; in our case there are two peers per Organization.

Here is a relevant portion of the crypto-config.yaml file.

|  |
| --- |
| OrdererOrgs:  #---------------------------------------------------------  # Orderer  # --------------------------------------------------------  - Name: Orderer  Domain: example.com  CA:  Country: US  Province: California  Locality: San Francisco  # OrganizationalUnit: Hyperledger Fabric  # StreetAddress: address for org # default nil  # PostalCode: postalCode for org # default nil  # ------------------------------------------------------  # "Specs" - See PeerOrgs below for complete description  # -----------------------------------------------------  Specs:  - Hostname: orderer  # -------------------------------------------------------  # "PeerOrgs" - Definition of organizations managing peer nodes  # ------------------------------------------------------  PeerOrgs:  # -----------------------------------------------------  # Org1  # ----------------------------------------------------  - Name: Org1  Domain: org1.example.com |

The naming convention for a network entity is as follows - “{{.Hostname}}.{{.Domain}}”. So using our ordering node as a reference point, we are left with an ordering node named - orderer.example.com that is tied to an MSP ID of Orderer. This file contains extensive documentation on the definitions and syntax.

After the cryptogen tool is executed, the generated certificates and keys will be saved to a folder titled crypto-config.

### Configuration Transaction Generator

The configtxgen tool is used to create four configuration artifacts:

1. The orderer genesis block,
2. The channel configuration transaction,
3. 3.two anchor peer transactions - one for each Peer Org.

The orderer block is the Genesis Block for the ordering service, and the channel transaction file is broadcast to the orderer at Channel creation time. The anchor peer transactions, as the name might suggest, specify each Org’s Anchor Peer on this channel.

## Lab 2. Running our first network

A script - script.sh - is baked inside the CLI container. The script drives the createChannel command against the supplied channel name and uses the channel.tx file for channel configuration.

The output of createChannel is a genesis block - <your\_channel\_name>.block - which gets stored on the peers’ file systems and contains the channel configuration specified from channel.tx.

The joinChannel command is exercised for all four peers, which takes as input the previously generated genesis block. This command instructs the peers to join <your\_channel\_name> and create a chain starting with <your\_channel\_name>.block.

Now we have a channel consisting of four peers, and two organizations. This is our TwoOrgsChannel profile.

peer0.org1.example.com and peer1.org1.example.com belong to Org1; peer0.org2.example.com and peer1.org2.example.com belong to Org2

These relationships are defined through the crypto-config.yaml and the MSP path is specified in our docker compose.

The anchor peers for Org1MSP (peer0.org1.example.com) and Org2MSP (peer0.org2.example.com) are then updated. We do this by passing the Org1MSPanchors.tx and Org2MSPanchors.tx artifacts to the ordering service along with the name of our channel.

A chaincode - chaincode\_example02 - is installed on peer0.org1.example.com and peer0.org2.example.com

The chaincode is then “instantiated” on peer0.org2.example.com. Instantiation adds the chaincode to the channel, starts the container for the target peer, and initializes the key value pairs associated with the chaincode. The initial values for this example are [“a”,”100” “b”,”200”]. This “instantiation” results in a container by the name of dev-peer0.org2.example.com-mycc-1.0 starting.

The instantiation also passes in an argument for the endorsement policy. The policy is defined as -P "OR ('Org1MSP.member','Org2MSP.member')", meaning that any transaction must be endorsed by a peer tied to Org1 or Org2.

A query against the value of “a” is issued to peer0.org1.example.com. The chaincode was previously installed on peer0.org1.example.com, so this will start a container for Org1 peer0 by the name of dev-peer0.org1.example.com-mycc-1.0. The result of the query is also returned. No write operations have occurred, so a query against “a” will still return a value of “100”.

An invoke is sent to peer0.org1.example.com to move “10” from “a” to “b”

The chaincode is then installed on peer1.org2.example.com

A query is sent to peer1.org2.example.com for the value of “a”. This starts a third chaincode container by the name of dev-peer1.org2.example.com-mycc-1.0. A value of 90 is returned, correctly reflecting the previous transaction during which the value for key “a” was modified by 10.

### Endorsement policies

Endorsement policies are used to instruct a peer on how to decide whether a transaction is properly endorsed. When a peer receives a transaction, it invokes the VSCC (Validation System Chaincode) associated with the transaction’s Chaincode as part of the transaction validation flow to determine the validity of the transaction. Recall that a transaction contains one or more endorsement from as many endorsing peers. VSCC is tasked to make the following determinations:

all endorsements are valid (i.e. they are valid signatures from valid certificates over the expected message)

there is an appropriate number of endorsements

endorsements come from the expected source(s)

Endorsement policies are a way of specifying the second and third points.

# The Hyperledger Fabric Model

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

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

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

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

Privacy through Channels - Channels enable multi-lateral transactions with the high degrees of privacy and confidentiality required by competing businesses and regulated industries that exchange assets on a common network.

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

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

## Assets

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

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

## Chaincode

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

## Ledger Features

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

The ledger is comprised of a blockchain (‘chain’) to store the immutable, sequenced record in blocks, as well as a state database to maintain current fabric state. There is one ledger per channel. Each peer maintains a copy of the ledger for each channel of which they are a member.

Query and update ledger using key-based lookups, range queries, and composite key queries

Read-only queries using a rich query language (if using CouchDB as state database)

Read-only history queries - Query ledger history for a key, enabling data provenance scenarios

Transactions consist of the versions of keys/values that were read in chaincode (read set) and keys/values that were written in chaincode (write set)

Transactions contain signatures of every endorsing peer and are submitted to ordering service

Transactions are ordered into blocks and are “delivered” from an ordering service to peers on a channel

Peers validate transactions against endorsement policies and enforce the policies

Prior to appending a block, a versioning check is performed to ensure that states for assets that were read have not changed since chaincode execution time

There is immutability once a transaction is validated and committed

A channel’s ledger contains a configuration block defining policies, access control lists, and other pertinent information

Channel’s contain Membership Service Provider instances allowing for crypto materials to be derived from different certificate authorities.

## Privacy through Channels

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

## Consensus

In distributed ledger technology, consensus has recently become synonymous with a specific algorithm, within a single function. However, consensus encompasses more than simply agreeing upon the order of transactions, and this differentiation is highlighted in Hyperledger Fabric through its fundamental role in the entire transaction flow, from proposal and endorsement, to ordering, validation and commitment. In a nutshell, consensus is defined as the full-circle verification of the correctness of a set of transactions comprising a block.

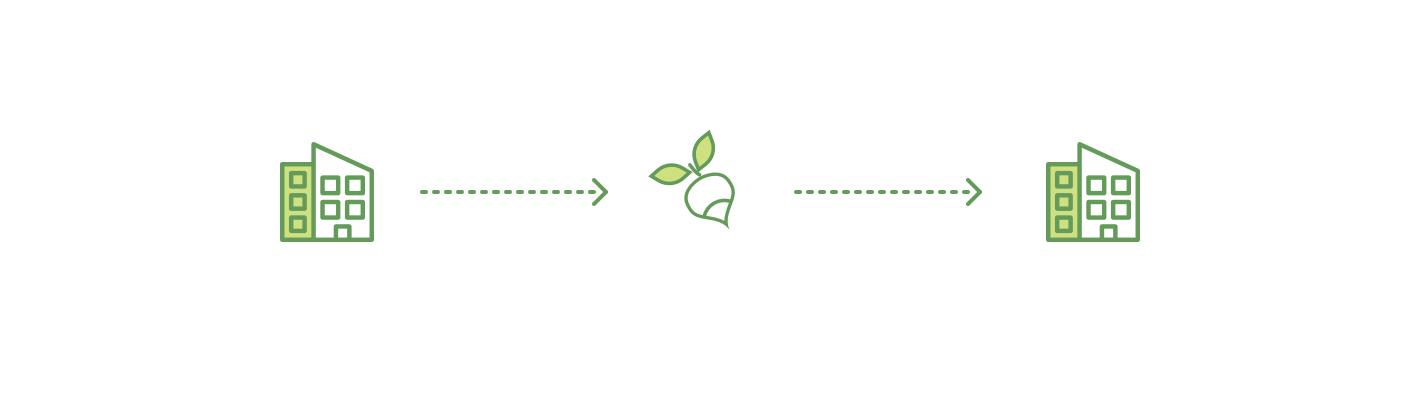
Consensus is ultimately achieved when the order and results of a block’s transactions have met the explicit policy criteria checks. These checks and balances take place during the lifecycle of a transaction, and include the usage of endorsement policies to dictate which specific members must endorse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

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

In the latter scenario, these participants would create a separate channel and thereby isolate/segregate their transactions and ledger. In order to solve scenarios that want to bridge the gap between total transparency and privacy, chaincode can be installed only on peers that need to access the asset states to perform reads and writes (in other words, if a chaincode is not installed on a peer, it will not be able to properly interface with the ledger). To further obfuscate the data, values within chaincode can be encrypted (in part or in total) using common cryptographic algorithms such as AES before appending to the ledger

## Fabric Transaction Flow

This document outlines the transactional mechanics that take place during a standard asset exchange. The scenario includes two clients, A and B, who are buying and selling radishes. They each have a peer on the network through which they send their transactions and interact with the ledger..



## Assumptions

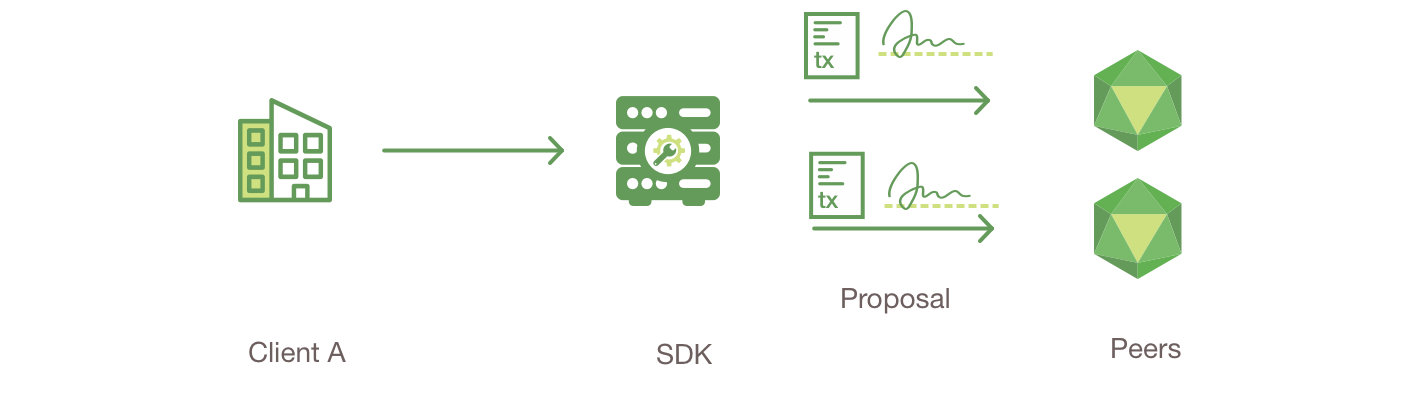
This flow assumes that a channel is set up and running. The application user has registered and enrolled with the organization’s certificate authority (CA) and received back necessary cryptographic material, which is used to authenticate to the network.

The chaincode (containing a set of key value pairs representing the initial state of the radish market) is installed on the peers and instantiated on the channel. The chaincode contains logic defining a set of transaction instructions and the agreed upon price for a radish. An endorsement policy has also been set for this chaincode, stating that both peerA and peerB must endorse any transaction.

## Client A initiates a transaction

What’s happening? - Client A is sending a request to purchase radishes. The request targets peerA and peerB, who are respectively representative of Client A and Client B. The endorsement policy states that both peers must endorse any transaction, therefore the request goes to peerA and peerB.

Next, the transaction proposal is constructed. An application leveraging a supported SDK (Node, Java, Python) utilizes one of the available API’s which generates a transaction proposal. The proposal is a request to invoke a chaincode function so that data can be read and/or written to the ledger (i.e. write new key value pairs for the assets). The SDK serves as a shim to package the transaction proposal into the properly architected format (protocol buffer over gRPC) and takes the user’s cryptographic credentials to produce a unique signature for this transaction proposal.



## Endorsing peers verify signature & execute the transaction

The endorsing peers verify

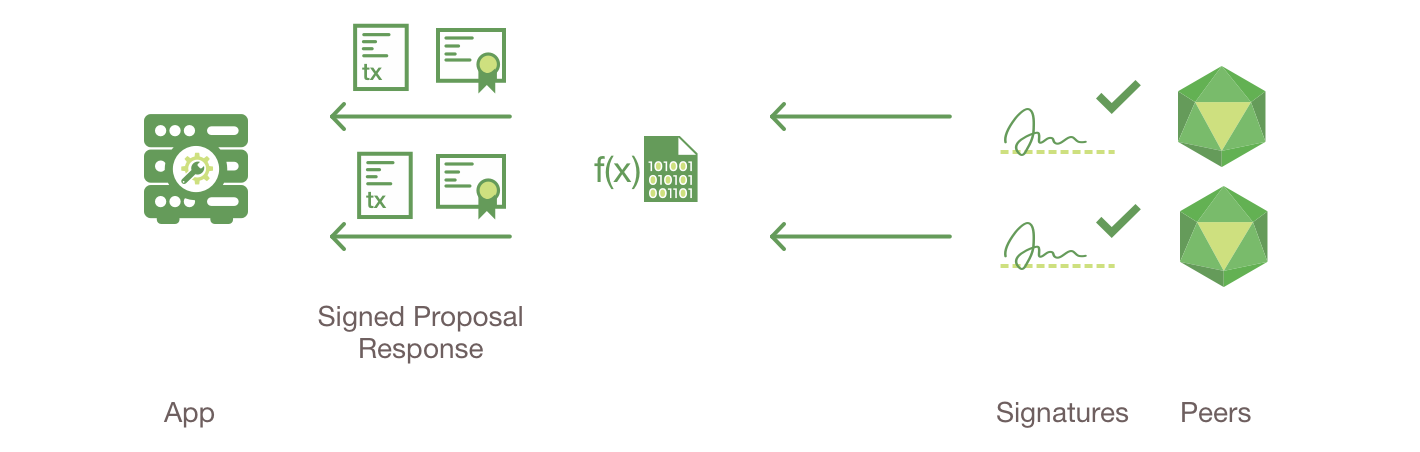
(1) that the transaction proposal is well formed,

(2) it has not been submitted already in the past (replay-attack protection),

(3) the signature is valid (using MSP), and

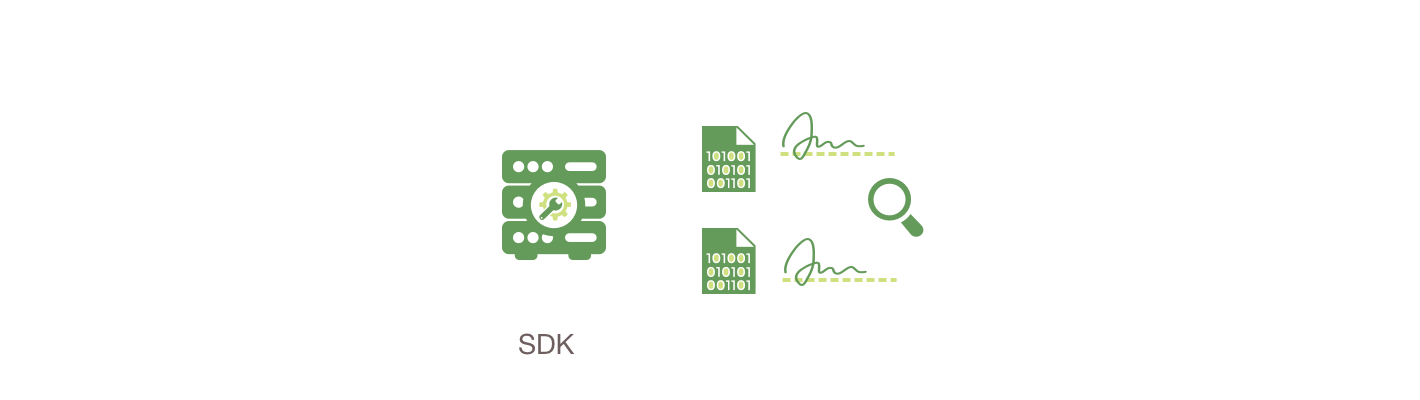
(4) that the submitter (Client A, in the example) is properly authorized to perform the proposed operation on that channel (namely, each endorsing peer ensures that the submitter satisfies the channel’s Writers policy).

The endorsing peers take the transaction proposal inputs as arguments to the invoked chaincode’s function. The chaincode is then executed against the current state database to produce transaction results including a response value, read set, and write set. No updates are made to the ledger at this point. The set of these values, along with the endorsing peer’s signature is passed back as a “proposal response” to the SDK which parses the payload for the application to consume.



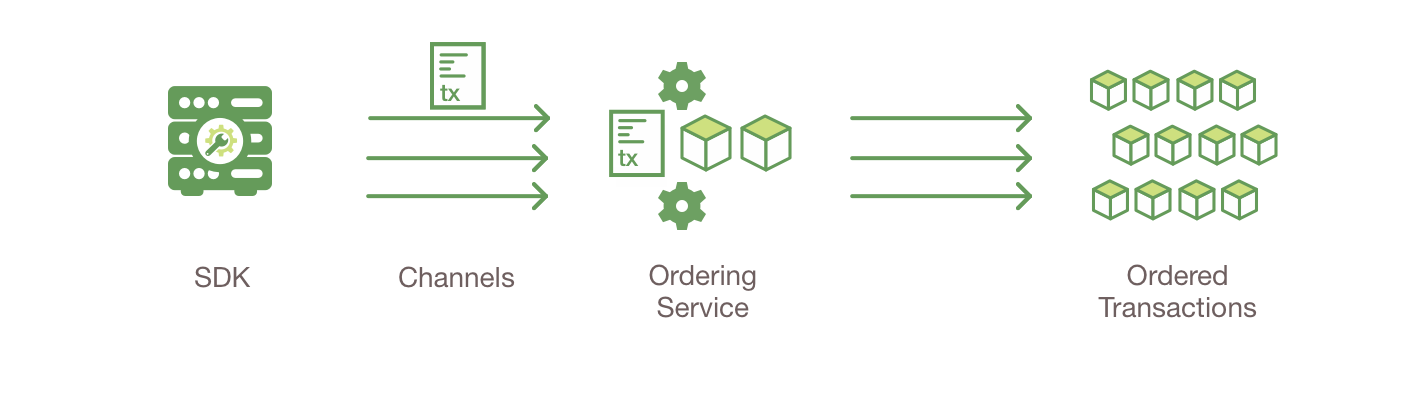
## Proposal responses are inspected

The application verifies the endorsing peer signatures and compares the proposal responses to determine if the proposal responses are the same. If the chaincode only queried the ledger, the application would inspect the query response and would typically not submit the transaction to Ordering Service. If the client application intends to submit the transaction to Ordering Service to update the ledger, the application determines if the specified endorsement policy has been fulfilled before submitting (i.e. did peerA and peerB both endorse). The architecture is such that even if an application chooses not to inspect responses or otherwise forwards an unendorsed transaction, the endorsement policy will still be enforced by peers and upheld at the commit validation phase.



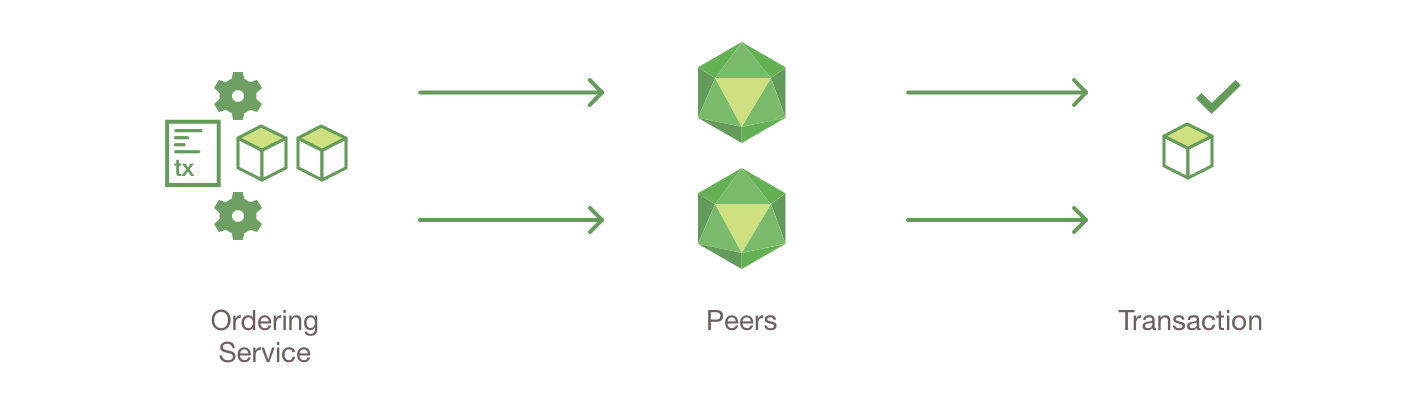
## Client assembles endorsements into a transaction

The application “broadcasts” the transaction proposal and response within a “transaction message” to the Ordering Service. The transaction will contain the read/write sets, the endorsing peers signatures and the Channel ID. The Ordering Service does not need to inspect the entire content of a transaction in order to perform its operation, it simply receives transactions from all channels in the network, orders them chronologically by channel, and creates blocks of transactions per channel.



## Transaction is validated and committed

The blocks of transactions are “delivered” to all peers on the channel. The transactions within the block are validated to ensure endorsement policy is fulfilled and to ensure that there have been no changes to ledger state for read set variables since the read set was generated by the transaction execution. Transactions in the block are tagged as being valid or invalid.



## Ledger updated

Each peer appends the block to the channel’s chain, and for each valid transaction the write sets are committed to current state database. An event is emitted, to notify the client application that the transaction (invocation) has been immutably appended to the chain, as well as notification of whether the transaction was validated or invalidated.

