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 Irona. 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 bulding 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.

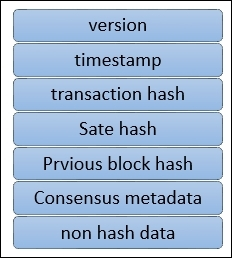
### The Consensus manager

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.

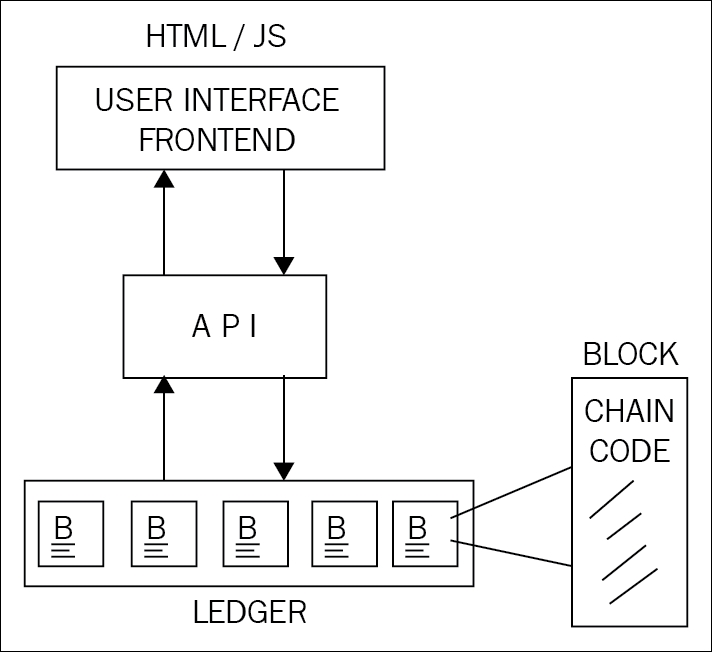
A non-validating peer does not execute transactions and only constructs transactions that are then forwarded to validating nodes.

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.