A HLF network is comprised of 3 main components:

- **Peer nodes**: execute chaincode, access ledger data, endorse transactions & interface with applications. They can be started, stopped, reconfigured & deleted at any point in the blockchain lifecycle.
- **Orderer nodes**: ensure the consistency of the blockchain and deliver the endorsed txns to the peers of the network. They are responsible for ordering txns, forming the ordering service, packaging txns into blocks & distributing these blocks to anchor pers across the network.
- **Membership Service Provider (MSP) nodes**: implemented as a **Certificate Authority** (CA), which manages X.509 certificates and are used to authenticate member identity and roles. These are the abstraction layer for all cryptographic mechanisms and protocols behind issuing certificates, validating certificates & authenticating users.

Practical Byzantine Fault Tolerance (PBFT) is the most used consensus algorithm in HLF.

HLF Architecture:

- Node
- **Client**: a node that can act on the end-users behalf.
- **Peers**: endorser and committer. **Endorsers** simulate & endorse txns. (A txn endorsement is a signed response to the results of the simulated txn.) **Committers** verify endorsements & validate txn results prior to committing txns to the blockchain.
- **Transactions: "Deploy txns"** and "**Invoke txns**". Deploy txns create a new chaincode with a parameter as a program and install the chaincode on the blockchain. Invoke txns execute in the context of a previous chaincode or smart contract deployment.
- **Data Structures:** Key-Value Store (KVS)-basic data structure of the blockchain. Keys: **names** & Values: **blobs**. KVS supports **PUT** & **GET**. Default db: LevelDB (can be replaced with CouchDB).
- **Ledger:**
- **Fabric Database:** state DB & ledger DB. The current state data represents the latest values for all assets in the ledger (also known as world state as it represents all the committed txns on the channel). The ledger DB stores serialized blocks where each block has 1 or more txns. Each txn contains a read-write set, which modifies 1 or more key/value pairs (definitive source of data & is immutable).

- **Ordering Mechanism:** HLF provides 3 ordering mechanisms:
  - **SOLO ordering mechanism:** involves a single ordering node (mostly used for experimenting).
  - **Kafka ordering mechanism:** recommended for production use. The data consists of endorsed txns & RW sets. Crash fault-tolerant as it uses Apache Kafka.
  - **Simplified Byzantine Fault Tolerance ordering mechanism:** is both crash fault-tolerant and byzantine fault-tolerant, meaning it can reach agreement even in the presence of malicious or faulty nodes.
- **Consensus:** process of reaching agreement on the next set of transactions to be added to the ledger. HLF is made of 3 steps:
  - First, in the txn endorsement step, an endorsing peer executes the chaincode, which, if it succeeds, yields an actual txn for the ledger. Then, the endorsing peer signs the txn and returns it to the proposer.
  - The ordering service supports strict ordering, meaning that if the Transaction T1 has already been written within block B1, then the same txn T1 cannot be re-written into different blocks like B2/B3.
  - Lastly, validation & commitment include txn validation & ledger commitment.
- **Smart Contract**

Prerequisites

To set up a development environment with Hyperledger Fabric, we'll need the following

1. The latest version of cURL
2. Docker and Docker Compose ≥v17.06.2-ce
3. Go v1.12.x
4. Node.js ≥v8.9 or ≥v10.15.3

Installations

- Curl
- Docker
- Go

- Node.js
- HLF

The **Fabric CA** component allows applications to enroll peers and application users to establish trusted identities on the blockchain network. It also provides support for pseudonym transaction submissions with Transaction Certificates.

The **Fabric Common** package encapsulates the common code used by all fabric-sdk-node packages, supporting fine-grained interactions with the Fabric network to send transaction invocations.

fabric-network is responsible for establishing a connection with the Fabric blockchain network. This package encapsulates the APIs to connect to a Fabric network, submit transactions, and perform queries against the ledger at a higher level of abstraction than through Fabric Common.

fabric-contract-api provides the contract interface where we write the business logic in technical words for our smart contracts or chaincode.

The **fabric-protos** package encapsulates the protobuffers that are used to communicate over gRPC.

The **Gateway** class is the entry point used to interact with a HLF blockchain network and the thing to be initiated. This provides a reusable connection to a peer within the Fabric blockchain network, enabling access to any of the blockchain networks or channels of which that particular peer is a member.

Using smart contract event listeners, client applications can initiate actions or business processes in response to chaincode events emitted by smart contract txns. All updates to the ledger can be observed using block event listeners.

Key chaincode APIs

The ChaincodeStub provides functions that allow developers to interact with the underlying ledger to query, update, and delete assets:

- `func (stub *ChaincodeStub) GetState(key string) ([]byte, error)`
  This returns the value of the specified key from the ledger. If the key doesn't exist in the state DB, nil, nil is returned.
- `func (stub *ChaincodeStub) PutState(key string, value []byte) error`
  This puts the specified key and value into the txn & writes it as a data-write proposal. PutState doesn't affect the ledger until the txn is validated and successfully committed.
- `func (stub *ChaincodeStub) DelState(key string) error`
  This records the specified key to be deleted in the Write set of the txn proposal. When the txn is validated & successfully committed, the key and its value will be deleted from the ledger.

You must create both an Init and an Invoke method within your chaincode. Before the chaincode can be invoked, the chaincode must be installed and instantiated using the peer chaincode install and instantiate commands, respectively.