**Introduction to Hyperledger Fabric Architecture**

Hyperledger Fabric is so unique, because it allows for modular consensus and membership service. This means that algorithms for consensus, identity verification are plug-and-play, resulting in a universal blockchain architecture, that can be applied to most industries or business models.

Channels are another unique feature. They allow transactions to be private between two actors, while still being verified and committed to the blockchain.

Roles within a Hyperledger Fabric Network

There are three different types of roles within a Hyperledger Fabric network:

1. *Clients*  
   Clients are applications that act on behalf of a person to propose transactions on the network.
2. *Peers*  
   Peers maintain the state of the network and a copy of the ledger.

There are two different types of peers:

* endorsing and
* committing peers.

However, there is an overlap between endorsing and committing peers, in that endorsing peers are a special kind of committing peers. All peers commit blocks to the distributed ledger.  
- Endorsers simulate and endorse transactions  
- Committers verify endorsements and validate transaction results, prior to committing transactions to the blockchain.

1. *Ordering Service*  
   The ordering service accepts endorsed transactions, orders them into a block, and delivers the blocks to the committing peers.

How to Reach Consensus

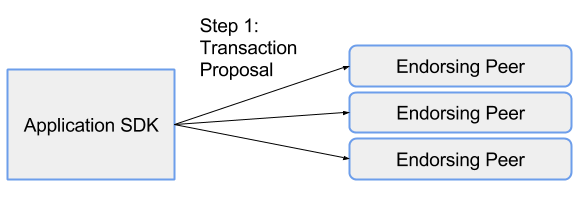
In a distributed ledger system, **consensus** is the process of reaching agreement on the next set of transactions to be added to the ledger.

Hyperledger Fabric, consensus is made up of three distinct steps:

* Transaction endorsement
* Ordering
* Validation and commitment.

Transaction Flow 1

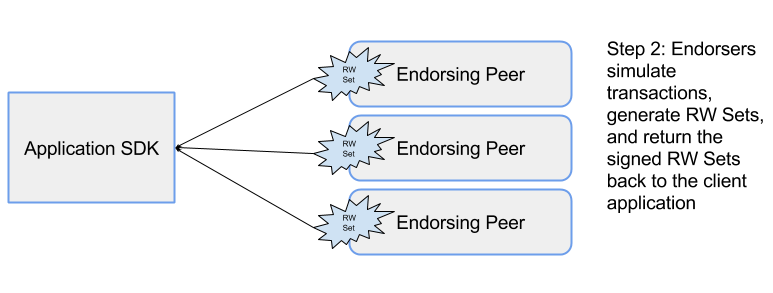
Within a Hyperledger Fabric network, transactions start out with client applications sending transaction proposals, or, in other words, proposing a transaction to endorsing peers.



**Client applications** are commonly referred to as **applications** or **clients**, and allow people to communicate with the blockchain network.

Transaction Flow 2

* Each endorsing peer simulates the proposed transaction, without updating the ledger.
* The endorsing peers will capture the set of **R**ead and **W**ritten data, called **RW Sets**.
* These RW sets are then signed by the endorsing peer, and returned to the client application to be used in future steps of the transaction flow.



Endorsing peers must hold smart contracts in order to simulate the transaction proposals.

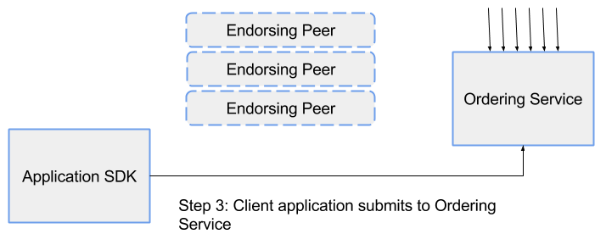
Transaction Endorsement

The method of transaction endorsements depends on the endorsement policy which is specified when the chaincode is deployed. An example of an endorsement policy would be "the majority of the endorsing peers must endorse the transaction".

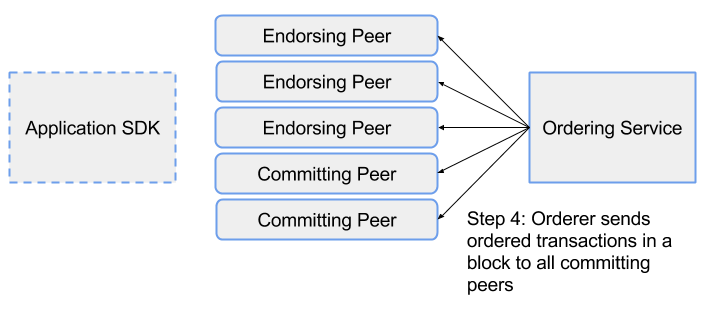
Since an endorsement policy is specified for a specific chaincode, different channels can have different endorsement policies.

Transaction Flow 3

The application then submits the endorsed transaction and the RW sets to the ordering service. Ordering happens across the network, in parallel with endorsed transactions and RW sets submitted by other applications.



Transaction Flow 4



The ordering service takes the endorsed transactions and RW sets, orders this information into a block, and delivers the block to all committing peers.

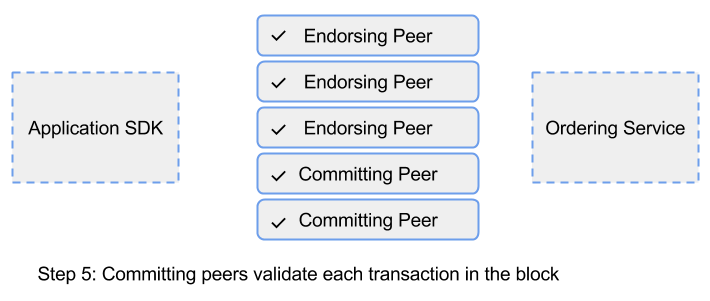
The **ordering service**, which is made up of a cluster of orderers, does not process transactions, smart contracts, or maintains the shared ledger. The ordering service accepts the endorsed transactions and specifies the order in which those transactions will be committed to the ledger.

The default ordering service for Hyperledger Fabric is Kafka.

Transaction Flow 5

The committing peer validates the transaction by checking to make sure that the RW sets still match the current world state.

When the committing peer validates the transaction, the transaction is written to the ledger, and the world state is updated with the Write data from the RW Set.

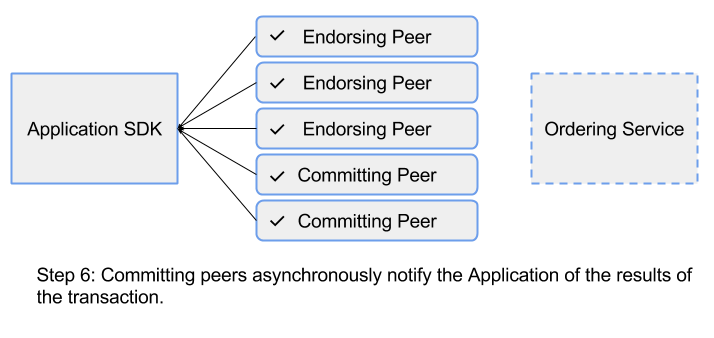


If the transaction fails, that is, if the committing peer finds that the RW set does not match the current world state, the transaction ordered into a block will still be included in that block, but it will be marked as invalid, and the world state will not be updated.

Committing peers are responsible for adding blocks of transactions to the shared ledger and updating the world state. They may hold smart contracts, but it is not a requirement.

Transaction Flow 6

Lastly, the committing peers asynchronously notify the client application of the success or failure of the transaction. Applications will be notified by each committing peer.

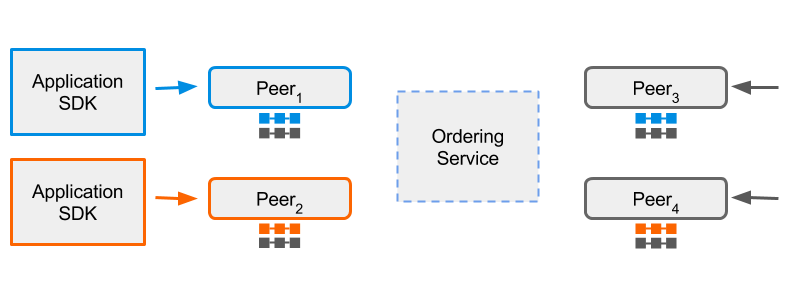


Identity Verification

In addition to the multitude of endorsement, validity, and versioning checks that take place, there are also ongoing identity verifications happening during each step of the transaction flow. Access control lists are implemented on the hierarchical layers of the network (from the ordering service down to channels), and payloads are repeatedly signed, verified, and authenticated as a transaction proposal passes through the different architectural components.

Channels

* Channels allow organizations to utilize the same network, while maintaining separation between multiple blockchains.
* Only the members of the channel on which the transaction was performed can see the specifics of the transaction.

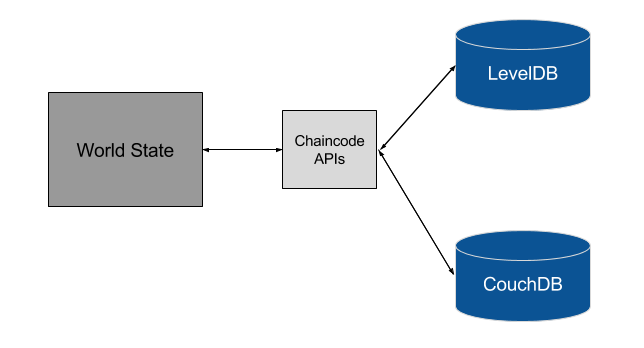


* The diagram above shows three distinct channels -- blue, orange, and grey. Each channel has its own application, ledger, and peers.
* Peers can belong to multiple networks or channels. Peers that do participate in multiple channels simulate and commit transactions to different ledgers.
* The ordering service is the same across any network or channel.

State Database

* The current state data represents the latest values for all assets in the ledger. Since the current state represents all the committed transactions on the channel, it is sometimes referred to as world state.
* Chaincode invocations execute transactions against the current state data.
* To make these chaincode interactions extremely efficient, the latest key/value pairs for each asset are stored in a state database.
* The state database will automatically get recovered (or generated, if needed) upon peer startup, before new transactions are accepted.

The default state database, **LevelDB**, can be replaced with **CouchDB**.



Smart Contracts

* Smart contracts are computer programs that contain logic to execute transactions and modify the state of the assets stored within the ledger.
* Hyperledger Fabric smart contracts are called chaincode and are written in Go.
* The chaincode serves as the business logic for a Hyperledger Fabric network, in that the chaincode directs how you manipulate assets within the network.

Membership Service Provider (MSP)

* The membership service provider, or MSP, is a component that defines the rules in which identities are validated, authenticated, and allowed access to a network.
* The MSP manages user IDs and authenticates clients who want to join the network.
* The MSP makes use of a *Certificate Authority*, which is a pluggable interface that verifies and revokes user certificates upon confirmed identity.
* The default interface used for the MSP is the **Fabric-CA API**, however, organizations can implement an External Certificate Authority of their choice.
* A single Hyperledger Fabric network can be controlled by multiple MSPs, where each organization brings their favorite.

What Does the MSP Do?

* Users are authenticated using a certificate authority.
* The certificate authority identifies the application, peer, endorser, and orderer identities, and verifies these credentials.
* A signature is generated through the use of a *Signing Algorithm* and a *Signature Verification Algorithm*.

Fabric-Certificate Authority

* *Certificate Authorities* manage enrollment certificates for a permissioned blockchain.
* **Fabric-CA** is the default certificate authority for Hyperledger Fabric, and handles the registration of user identities.
* The Fabric-CA certificate authority is in charge of issuing and revoking Enrollment Certificates (E-Certs).
* The current implementation of Fabric-CA only issues E-Certs, which supply long term identity certificates. E-Certs, which are issued by the Enrollment Certificate Authority (E-CA), assign peers their identity and give them permission to join the network and submit transactions.

Fabric-CA Features

* registration of identities, or connects to LDAP as the user registry
* issuance of Enrollment Certificates (ECerts)
* certificate renewal and revocation

Where Fabric-CA are stored

<http://hyperledger-fabric-ca.readthedocs.io/en/latest/users-guide.html>

Hyperledger Fabric CA consists of both a server and a client component.

_images/fabric-ca.png

* There are two ways of interacting with a Hyperledger Fabric CA server: via the Hyperledger Fabric CA client or through one of the Fabric SDKs.
* All communication to the Hyperledger Fabric CA server is via REST APIs.

How to Setup the Hyperledger Fabric-ca Client Setup on Ubuntu 16.04

<https://mlgblockchain.com/setup-hyperledger-client.html>

Technical Prerequisites

In order to successfully install Hyperledger Fabric, you should be familiar with Go and Node.js programming languages, and have the following features installed on your computer: cURL, Node.js, npm package manager, Go language, Docker, and Docker Compose.

o run Hyperledger Composer and Hyperledger Fabric, we recommend you have at least 4Gb of memory.

The following are prerequisites for installing the required development tools:

* Operating Systems: Ubuntu Linux 14.04 / 16.04 LTS (both 64-bit), or Mac OS 10.12
* Docker Engine: Version 17.03 or higher
* Docker-Compose: Version 1.8 or higher
* Node: 8.9 or higher (note version 9 is not supported)
* npm: v5.x
* git: 2.9.x or higher
* Python: 2.7.x
* A code editor of your choice, we recommend VSCode.

\*\*If installing Hyperledger Composer using Linux, be aware of the following advice:

* Login as a normal user, rather than root.
* Do not su to root.
* When installing prerequisites, use curl, then unzip using sudo.
* Run prereqs-ubuntu.sh as a normal user. It may prompt for root password as some of it's actions are required to be run as root.
* Do not use npm with sudo or su to root to use it.
* Avoid installing node globally as root.\*\*

If you're running on Ubuntu, you can download the prerequisites using the following commands:

Copy

curl -O https://hyperledger.github.io/composer/latest/prereqs-ubuntu.sh

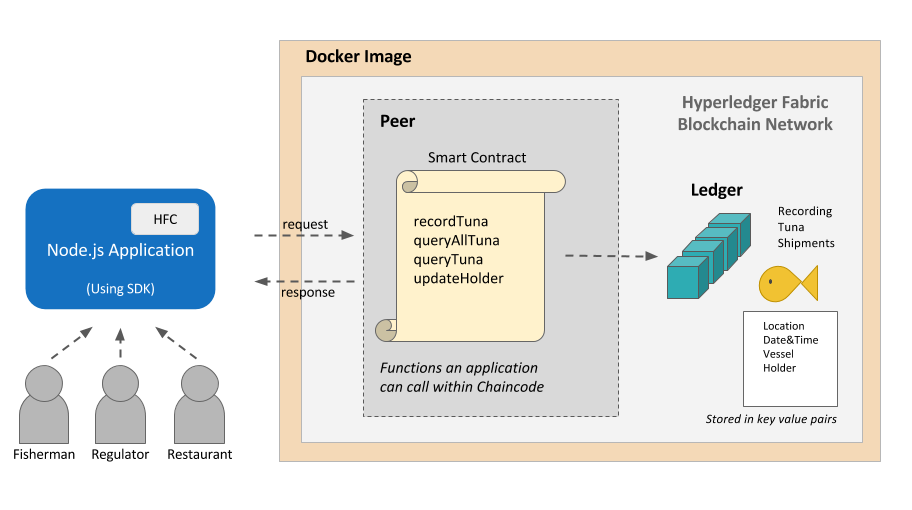
chmod u+x prereqs-ubuntu.sh

Next run the script - as this briefly uses sudo during its execution, you will be prompted for your password.

Copy

./prereqs-ubuntu.sh

Application Flow Example



1. Various users (fisherman, regulators, or restaurateurs etc.) will interact with the Node.js application.
2. The client JS will send messages to the backend when the user interacts with the application.
3. Reading or writing the ledger is known as a proposal (for example, querying a specific Tuna catch - queryTuna-  or recording a tuna catch - recordTuna). This proposal is built by our application via the SDK, and then sent to the endorsing peers.
4. The endorsing peers will use the application-specific chaincode smart contract to simulate the transaction. If there are no issues, the transaction will be endorsed, and sent back to our application.
5. Our application will then send the endorsed proposal to the ordering service via the SDK. The orderer will package many proposals from the whole network into a block. Then, it will broadcast the new block to the committing peers in the network.
6. Finally, each committing peer will validate the block and write it to its ledger (shown in teal above). The transaction has now been committed, and any reads will reflect this change.