**Hyperledger Intern**

**Fabric & Sawtooth**

**Smart Contract analysis**

**謝孟剛(Roger Hsieh)**

Table of Contents

[一、 Fabric、Sawtooth introduction 3](#_Toc530067839)

[1. Hyperledger Fabric 3](#_Toc530067840)

[2. Hyperledger Sawtooth 6](#_Toc530067841)

[二、 Smart Contract, Chaincode and Transaction Family 11](#_Toc530067842)

[1. Smart Contract 11](#_Toc530067843)

[2. Fabric Chaincode (Smart Contract) 11](#_Toc530067844)

[3. Sawtooth Transaction Family (Smart Contract) 13](#_Toc530067845)

[4. Transaction Processor & Chaincode 17](#_Toc530067846)

[5. Performance 17](#_Toc530067847)

[6. Security 17](#_Toc530067848)

[7. Mechanism 18](#_Toc530067849)

[三、 What’s the difference about implementing supply chain 18](#_Toc530067850)

[1. For example, develop material supply chain with Hyperledger Fabric.(By myself) 18](#_Toc530067851)

[2. Tuna supply chain with Hyperledger Sawtooth(code source: https://github.com/hyperledger/education/tree/master/LFS171x/sawtooth-material/sawtooth-tuna) 21](#_Toc530067852)

[3. The comparison in development supply chain application between Fabric and Sawtooth 26](#_Toc530067853)

[四、 Real Applications 27](#_Toc530067854)

[1. Real Applications on Fabric 27](#_Toc530067855)

[2. Real Applications on Sawtooth 28](#_Toc530067856)

[五、 Resource 29](#_Toc530067857)

1. Fabric、Sawtooth introduction
   1. Hyperledger Fabric
      1. Fabric is the first project join the Hyperledger. Fabric is contributed by IBM, DAH … and so on in 2015. Github link is <https://github.com/hyperledger/fabric>。
      2. Fabric let the function of ordering be an independent node. Reduce the burden of Peer node. Elimnate network processing bottle neck and improve scalability. There are two kinds of peer, Endorser and Committer. We can deploy different nodes according to loading and performance. Independent Fabric CA provide Publick Key Infrastructure and manage identity and certificate of Fabric network. Users get legal certificate through Client from Fabric CA server, then users can join the Fabric network and launch transaction.

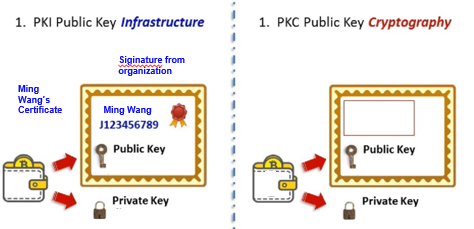
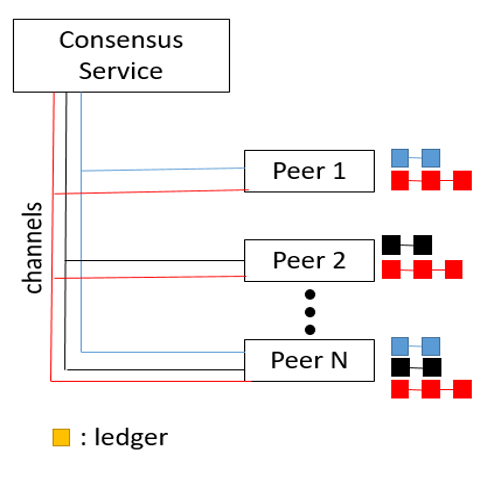


Figure: Compare PKI and PKC

(Source : 杜宏毅Blockchain的前世今生與未來)

* + 1. Implement multiple Channel mechanism. Isolate information from different channel. Users have the information only about the transaction they participated. Accomplish the commercial demand.

 Figure: Channel

( Data source : draw by myself，reference : http://www.gooread.com/article/20120676648/)

* + 1. Chaincode is the key concept of Fabric. Including Application Chaincode and System Chaincode. Application Chaincode is deployed on independent container. Through programming Application Chaincode change the state in the Ledger. System Chaincode is embedded in Fabric. We can manage Fabric network through System Chaincode.
    2. There are gRPC API and SDK for client to call. Applications can access many resources on Fabric network through SDK, including Ledger, Transaction, Chaincode, Event, and CA … and so on. Ledger is the most important. Ledger records application messages. Applications launch transaction to record data in Ledger. The transaction process logic is implemented by Chaincode. All events in Fabric network can be accessed by applications and trigger other procedure. Through CA we can manage the access controlling. There are so many technology mechanism in Fabric, including database, consensus, Container, World State, PKI, certificate, hash … and so on. Fabric network is peer to peer network and communicate with gRPC and Gossip.

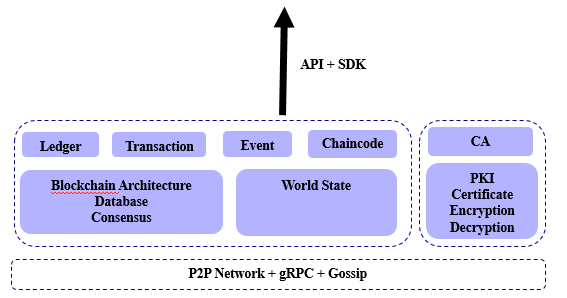
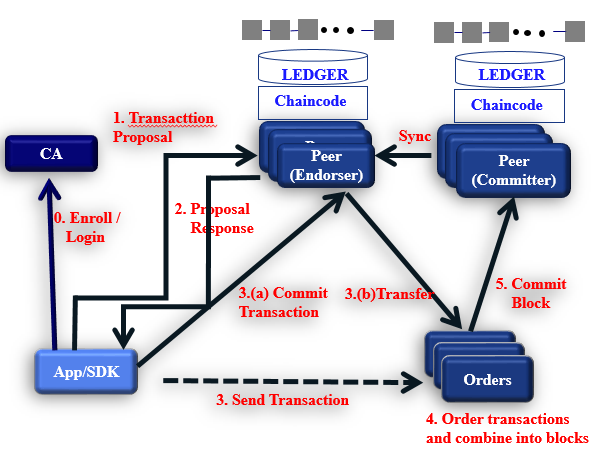


Figure: Fabric Architecture

(Data source: draw by myself, reference : https://read01.com/zh-tw/AJxmMKm.html#.WrCcC-huZPb)

Fabric Transaction Process.

Applications communicate with Fabric network through Hyperledger Fabric SDK. At first Applications get legal certificate from CA, then applications can join application channel. Step 1: Applications(Clients) send transaction proposal to Endorser peer. Step 2: Endorser will execute legally check and Access Control List Authorization check. If the checking has passed, Endorser will simulate transaction through Chaincode. Record the state with read/write set with endorsement and then return to Applications. Step 3: While Applications receive enough endorsements, it means the transaction is a legal request and then send read/write set to Orders. Step 4: Orders execute global ordering to legal trnsactions with timestamp. After the transactions had been ordered, the transactions will combine to form a block. Step 5: Then Orders transfer the block to Committer. Committer execute the final check. After it passes the check, Committer will execute legal transaction and record the result into Ledger and then synchronize to other Peer.

Figure: Fabric Transaction Process

* 1. Hyperledger Sawtooth
     1. Hyperledger Sawtooth is an enterprise blockchain platform for building distributed ledger applications and networks. The design philosophy targets keeping ledgers distributed and making smart contracts safe, particularly for enterprise use.

Sawtooth simplifies blockchain application development by separating the core system from the application domain. Application developers can specify the business rules appropriate for their application, using the language of their choice, without needing to know the underlying design of the core system.

Sawtooth is also highly modular. This modularity enables enterprises and consortia to make policy decisions that they are best equipped to make. Sawtooth’s core design allows applications to choose the transaction rules, permissioning, and consensus algorithms that support their unique business needs.

* + 1. Sawtooth makes it easy to develop and deploy an application by providing a clear separation between the application level and the core system level. Sawtooth provides smart contract abstraction that allows application developers to write contract logic in a language of their choice. Transaction processor SDKs are available in multiple languages to streamline creation of new contract languages, including Python, JavaScript, Go, C++, Java, and Rust. A provided REST API simplifies client development by adapting validator communication to standard HTTP/JSON.
    2. Sawtooth is built to solve the challenges of permissioned (private) networks. Clusters of Sawtooth nodes can be easily deployed with separate permissioning. There is no centralized service that could potentially leak transaction patterns or other confidential information.

The blockchain stores the settings that specify the permissions, such as roles and identities, so that all participants in the network can access this information.Parallel Transaction Execution

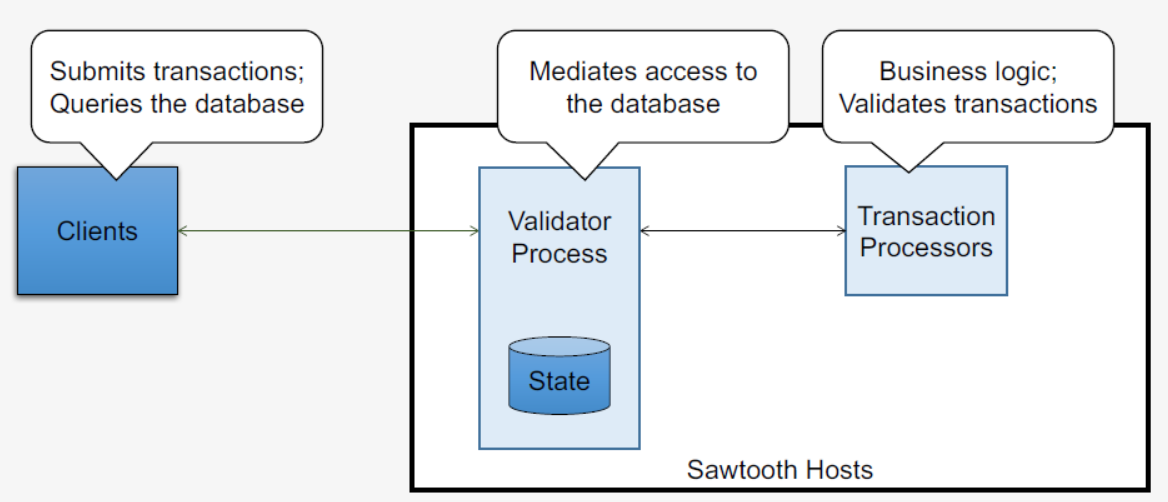
* + 1. Most blockchains require serial transaction execution in order to guarantee consistent ordering at each node on the network. Sawtooth includes an advanced parallel scheduler that splits transactions into parallel flows. Based on the locations in state which are accessed by a transaction, Sawtooth isolates the execution of transactions from one another while maintaining contextual changes. When possible, transactions are executed in parallel, while preventing double-spending even with multiple modifications to the same state. Parallel scheduling provides a substantial potential increase in performance over serial execution.
    2. Sawtooth Architecture

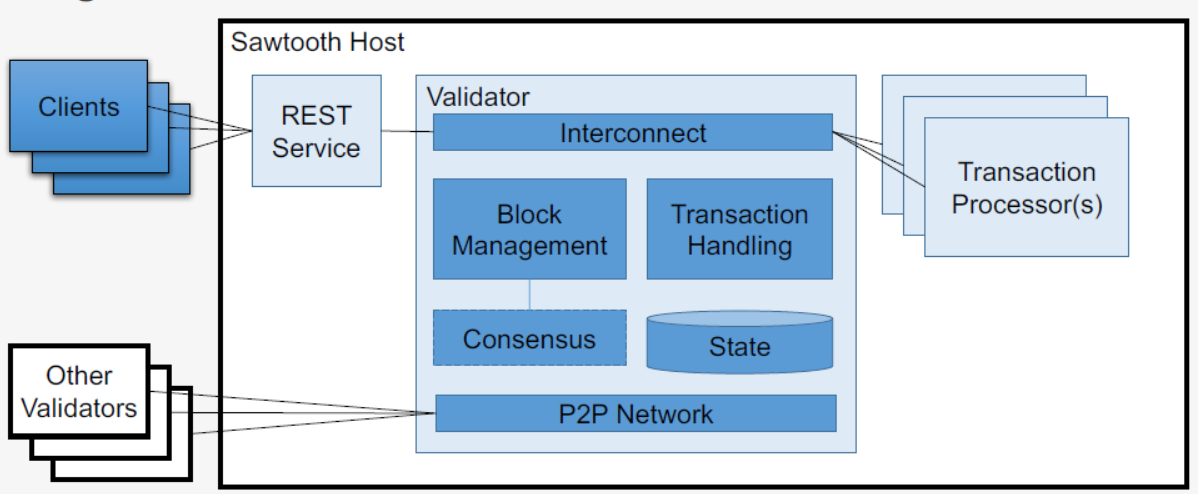
Figure: Sawtooth Architecture

Client applications send transactions to the blockchain, typically through the Sawtooth REST API. Provides a user interface for the application. A client can be a command-line interface, a web page, a mobile app, an IoT sensor, or most any other kind of interface capable of sending HTTP requests.

Sawtooth validators validate transactions, combine batches of transactions into blocks, submit them to the ledger, and approve valid blocks according to the network's consensus algorithm.

Sawtooth applications are distributed applications (also called transaction families) that consist of a transaction processor for the server-side logic and a client for use from Web, CLI, or mobile applications.

Transaction processors provide the server-side business logic.

Figure: Sawtooth Architecture

* 1. Sawtooth V.S. Fabric

Hyperledger Fabric and Hyperledger Sawtooth both are Distributed Ledger Technology (DLT) solutions, both backed by bigger technology players, like Hyperledger Fabric is under Linux Foundations and Hyperledger Sawtooth is under Intel. Even though both are drilling down the same path, there are some major differences that put these two far apart from each other.

Hyperledger Fabric is more focused on fine grained governance over participation in the network by MemberShip Provider (MSP) services, an MSP identifies which Certificate Authorities are trusted to define a member of an organization, which means to be a part of the network one must have to come from a trusted CA and so the Fabric Network are permissioned network and have no support for a permissionless network. Whereas, the Hyperledger Sawtooth provide a DLT network which can be either be Permissioned or permissionless. Sawtooth defines roles and permissions for achieving much more flexible approach than Hyperledger Fabric and the authentication is handled by the combination of large number of validators in the network and a novel consensus algorithm PoET (Proof of Elapsed Time).

* + 1. Validators V.S. Endorsement Peers

The process of validating a transaction in Fabric involves a set of endorsing peers defined by the endorsement policy. After validation, the distribution of the transaction is handled by ordering services.

However in the Sawtooth architecture, there are validators which take care of validating the transactions in the network as well as distributing the transaction to other peers.

* + 1. The State store

Each framework handles maintaining the state of the blockchain network in its own way.

Hyperledger Fabric stores the data in either leveldb or couchdb based on the setup and manages a ledger per channel.

In the case of Sawtooth, all the data is stored in a specific address and this address is generated based on the corresponding transaction processor’s prefix. For handling rich queries an instance of rethinkdb is setup and the ledger is replicated and synced with rethinkdb.

* + 1. Component Topology

Hyperledger Fabric has a lot of components which function together to form a blockchain network. This includes Orderers, Peers, CAs, CouchDB and Tools.

But in the case of Sawtooth there is only three major components which are Sawtooth Validator, Transaction Processors and the REST API for the Transaction Processors. This reduces the complexity of the network and brings more scalability.

* + 1. Consensus Algorithm

Hyperledger Fabric supports Kafka based consensus, whereas, Hyperledger Sawtooth supports PoET consensus algorithm.

* + 1. Scalability

While Hyperledger Fabric if more and more restricted in nature for participation, the Hyperledger Sawtooth is more flexible in this concern as it can be made open to join, blockchain network supported by Sawtooth can grow very large.

* + 1. Hyperledger Fabric Channel

Hyperledger Fabric have unique concept of channels which provide fine grained privacy controls over transactions, which best suits a B2B scenarios as well as many other scenarios where its application can be found.

* + 1. Hyperledger Fabirc MSP and CA

Using MSP and Different CA, Fabric has a better and tighter governance framework.

* + 1. Language support

Hyperledger supports Chaincode those can be written in either go, java and JS as well but they more or less behave like an asset based smart contracts. Whereas Sawtooth have transaction families that can be written in various languages whichever suits the developer and using them the smart contracts can be simulated in that languages, however sawtooth itself provides some out of the box transaction families it opens up window for developing many more. Sawtooth also have support for Ethereum Solidity based smart contracts with its Seth Framework which is not supported by Fabric at all.

* + 1. Sawtooth v.s. Fabric

|  |  |  |
| --- | --- | --- |
| Topic | Sawtooth | Fabric |
| Mode of operation | Permissioned | Permissioned or private |
| Consensus Algorithm | Proof of Elapsed Time | Allow users to choose the consensus algorithm |
| Smart Contract | Go, java, Python, javascript, Rust, C++, Solidity | Go, java, javascript |
| Currency | None | None or Currency or Token via chaincode |
| Governance | Linux Foundation | Linux Foundation |
|  | Focuses on creating very secure way to handle your smart contracts, with stricter rules and consensus. | Can optionally enforce these rules, which helps a lot of different private blockchain providers right now. |
| BFT enabled | Sawtooth is Enterprise ready, the major reason is that it comes with Byzantine Fault Tolerance features. | This BFT feature gives much higher tolerance rate than the Crash Fault Tolerance in Fabric |
| Parallel Transactions | The ability to have parallel transactions with Sawtooth makes all the difference.  This reducing the amount of time it takes to process transactions in the system. |  |
| Distributed ledger | Sawtooth aims to have distributed ledgers. |  |

1. Smart Contract, Chaincode and Transaction Family
   1. Smart Contract

Smart Contract is published by Nick Szabo in 1995. He mentioned the philosophy of Smart Contract on his own website. His definition about Smart Contract is that a Smart Contract is a numerical promise. Ensure participants will execute these promise.

Although the meaning of Smart Contract is not confirmed in law, Smart Contract is a computational agreement that once it deployed, it will self-execute and self-verify without artificial control.

* 1. Fabric Chaincode (Smart Contract)
     1. System Chaincode、Application Chaincode

There are two kinds of Chaincode in Fabric, including System Chaincode and Application Chaincode. System Chaincode is about system configuration, like endorsement, policy and validation … and so on. Application Chaincode is about server side logic.

* + 1. Programming Application Chaincode

We can use many language to program Chaincode, including Go, java and javascript. I use javascript here. At first we import fabric-shim package, then we can use it to write Chaincode. Declare Chaincode to a class, use shim.start to start the Chaincode at last. There are init and invoke function in every Chaincode. Init is the function will be automatic executed when Chaincode instantiation. Invoke is the entry point of Chaincode when we communicate with Chaincode. At first we call invoke function, then invoke function will call another function.

'use strict';

const shim = require('fabric-shim');

const util = require('util');

let Chaincode = class {

// The Init method is called when the Chaincode is instantiated by the blockchain network

async Init(stub) {

console.info('=========== Instantiated fabcar chaincode ===========');

return shim.success();

}

async Invoke(stub) {

let ret = stub.getFunctionAndParameters();

console.info(ret);

let method = this[ret.fcn];

if (!method) {

console.error('no function of name:' + ret.fcn + ' found');

throw new Error('Received unknown function ' + ret.fcn + ' invocation');

}

try {

let payload = await method(stub, ret.params, this);

return shim.success(payload);

} catch (err) {

console.log(err);

return shim.error(err);

}

}

//query ledger by its key

async queryByKey(stub, args, thisClass) {

if (args.length != 1) {

throw new Error('Incorrect number of arguments. Expecting CarNumber ex: CAR1');

}

let keyNumber = args[0];

let vlaueAsBytes = await stub.getState(keyNumber); //get the key's value from chaincode state

if (!valueAsBytes || vlaueAsBytes.toString().length <= 0) {

throw new Error(keyNumber + ' does not exist: ');

}

console.log(valueAsBytes.toString());

return valueAsBytes;

}

//create data into ledger

async createData(stub, args, thisClass) {

console.info('============= START : Create data ===========');

if (args.length != 7) {

throw new Error('Incorrect number of arguments. Expecting 7');

}

var data = {

docType: 'Data',

name: args[1],

product: args[2],

frequency: args[3],

interval: args[4],

ownerID: args[5],

timeStamp: args[6]

};

await stub.putState(args[0], Buffer.from(JSON.stringify(data)));

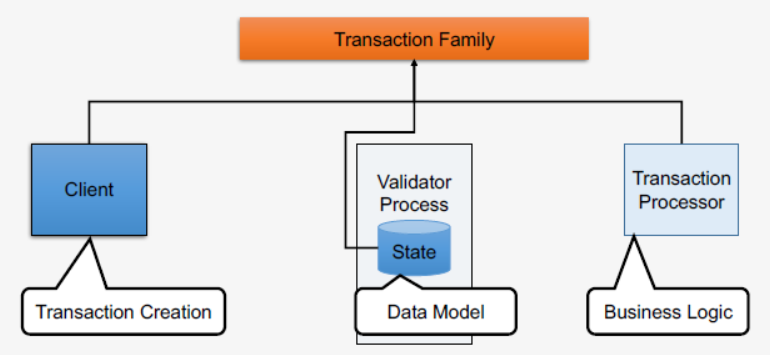
console.info('============= END : Create data ===========');

}

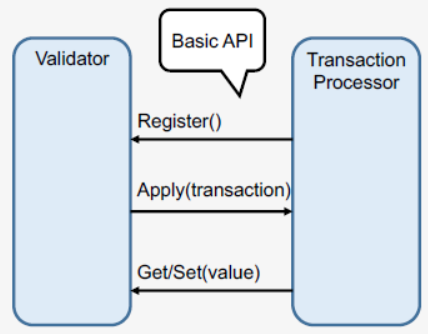
};

shim.start(new Chaincode());

There is partial example about Chaincode above.

* 1. Sawtooth Transaction Family (Smart Contract)
     1. Transaction Processor

Transaction Families encapsulate business logic on Sawtooth. A Transaction Family can be as simple as a single transaction format, with associated validity and state update logic or as complex as a VM with opcode accounting and bytecode stored in state. The choice is up to the developer.



All validators in the network run every authorized transaction processor. On receipt of a transaction the validator will call the Transaction Processor’s Apply() method. Business logic simply goes in Apply()

* + 1. The Client

Clients can be browser apps, CLIs, etc. Main job is to package and sign transactions & batches. Clients can post batches through the Rest API or connect to the validator directly.

* + 1. The Data Model

Both Client and Transaction Processor must use the same.

Data model, Serialization / encoding and Addressing scheme.

* + 1. About writing transaction processor

A transaction processor has two top-level components:

• Processor class. The SDK provides a general-purpose processor class.

• Handler class. The handler class is application-dependent. It contains the business logic for a particular family of transactions. Multiple handlers can be connected to an instance of the processor class.

Entry Point

Since a transaction processor is a long running process, it must have an entry point.

In the entry point, the TransactionProcessor class is given the address to connect with the validator and the handler class.

a simplified sawtooth\_xo/processor/main.py

from sawtooth\_sdk.processor.core import TransactionProcessor

from sawtooth\_xo.processor.handler import XoTransactionHandler

def main():

# In docker, the url would be the validator's container name with

# port 4004

processor = TransactionProcessor(url='tcp://127.0.0.1:4004')

handler = XoTransactionHandler()

processor.add\_handler(handler)

processor.start()

Handlers get called in two ways: with an apply method and with various “metadata” methods. The metadata is used to connect the handler to the processor. The bulk of the handler, however, is made up of apply and its helper functions.

sawtooth\_xo/processor/handler.py XoTransactionHandler class

class XoTransactionHandler(TransactionHandler):

def \_\_init\_\_(self, namespace\_prefix):

self.\_namespace\_prefix = namespace\_prefix

@property

def family\_name(self):

return 'xo'

@property

def family\_versions(self):

return ['1.0']

@property

def namespaces(self):

return [self.\_namespace\_prefix]

def apply(self, transaction, context):

# ...

Note that the XoTransactionHandler extends the TransactionHandler defined in the Python SDK.

The apply Method

apply gets called with two arguments, transaction and context. The argument transaction is an instance of the class Transaction that is created from the protobuf definition. Also, context is an instance of the class Context from the python SDK.

transaction holds the command that is to be executed (e.g. taking a space or creating a game), while context stores information about the current state of the game (e.g. the board layout and whose turn it is).

The transaction contains payload bytes that are opaque to the validator core, and transaction family specific. When implementing a transaction handler the binary serialization protocol is up to the implementer.

To separate details of state encoding and payload handling from validation logic, the XO example has XoState and XoPayload classes. The XoPayload has name, action, and space fields, while the XoState contains information about the game name, board, state, and which players are playing in the game.

Valid actions are: create a new game, take an unoccupied space, and delete a game.

sawtooth\_xo/processor/handler.py apply overview

def apply(self, transaction, context):

header = transaction.header

signer = header.signer\_public\_key

xo\_payload = XoPayload.from\_bytes(transaction.payload)

xo\_state = XoState(context)

if xo\_payload.action == 'delete':

...

elif xo.payload.action == 'create':

...

elif xo.payload.action == 'take':

...

else:

raise InvalidTransaction('Unhandled action: {}'.format(

xo\_payload.action))

For every new payload, the transaction processor validates rules surrounding the action. If all of the rules validate, then state is updated based on whether we are creating a game, deleting a game, or updating the game by taking a space.

* 1. Transaction Processor & Chaincode

One of the main advantages of Sawtooth is that the “smart contract” or transaction processors (as called in the Sawtooth world) can be written in wide verity of languages like Rust, Python or Javascript. Migrating transaction processors to the chain is also different compared to Fabric.

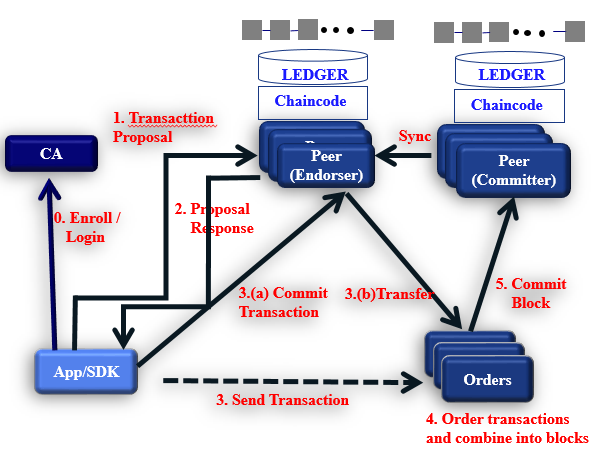
For now, there are more Fabric Chaincode examples and rich SDK document for Fabric. And there are more functions of SDK for Fabric. Take query for example, there are many kinds of query, including by range, by key, by composite key, rich query, GetPrivateData. There are less examples and documents for Sawtooth Transaction Processor. And the function of SDK is not rich enough. But sure, there are advantages for Transaction Family. The languages for transaction family is more than Fabric. We can set identity with Transaction Family. It’s easy than Fabric with CA.

* 1. Performance
     1. TPS
     2. The velocity to combine block
     3. The velocity of communication
     4. The velocity of document、program updating
  2. Security

Transaction Family is safer than Chaincode. In Transaction Family Client has to communicate with Validator through REST API and then Validator will communicate with certificated Transaction Processor. It protects the server side logic well. But In Fabric Client can communicate with Chaincode directly after client is certificated by CA.

* 1. Mechanism

There is a unique characteristic named channel in Fabric. Because of it separates peer into Endorser and Committer. Let different peer can do different jobs.

Figure: Fabric Transaction Architecture

Sawtooth is more light. Procedures like block, transaction, consensus are on Validator.

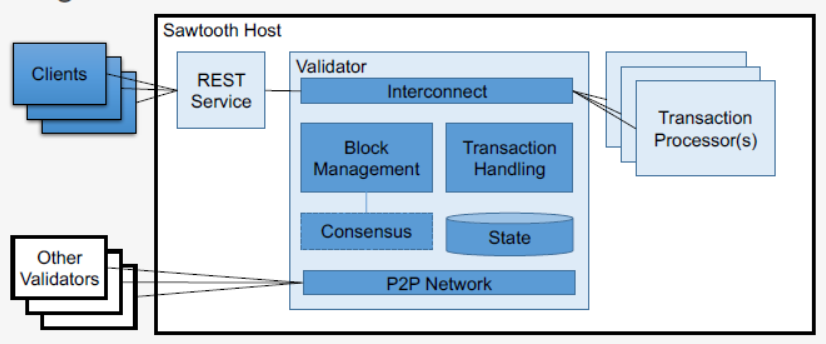


Figure: Sawtooth Architecture

1. What’s the difference about implementing supply chain
   1. For example, develop material supply chain with Hyperledger Fabric.(By myself)
      1. Introduction

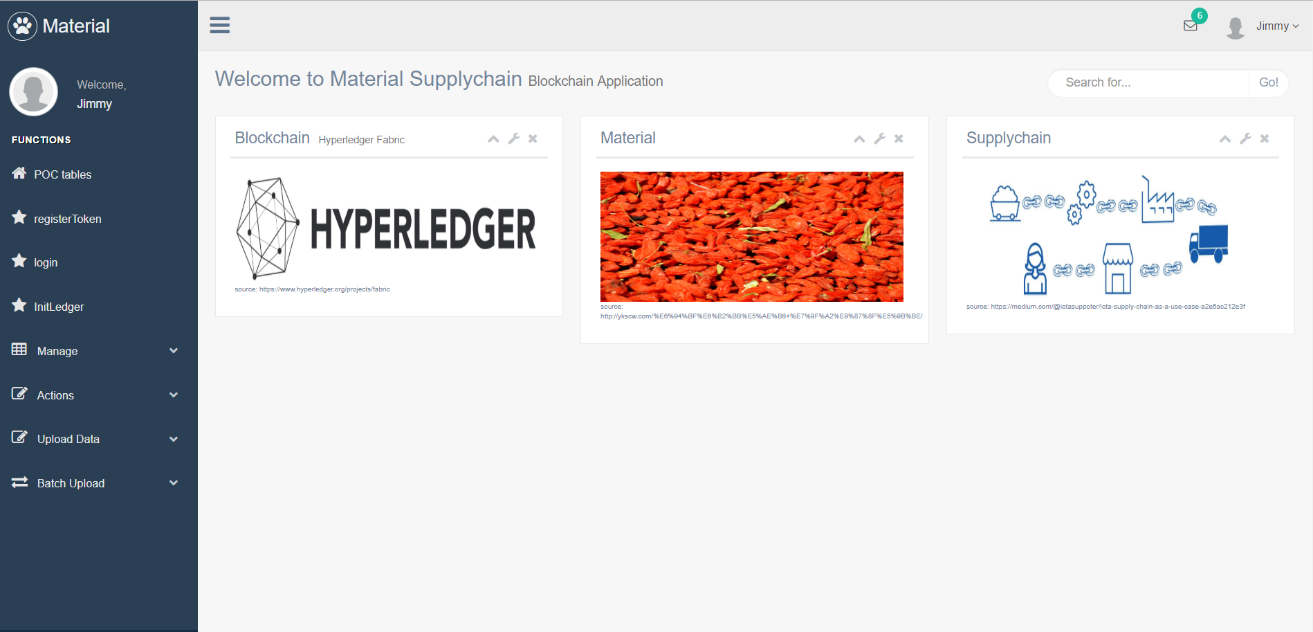
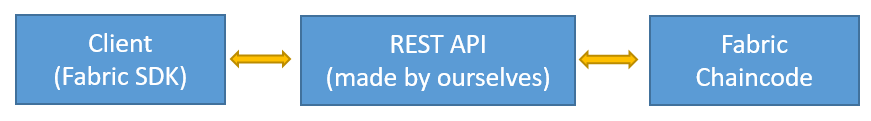
Implement material supply chain with Hyperledger Fabric to increase security. Store data to blockchain from farmer, plant base, wholesaler, factory, transportation, drugstore to hospital. Real time data will be on chain and accomplish immutability.

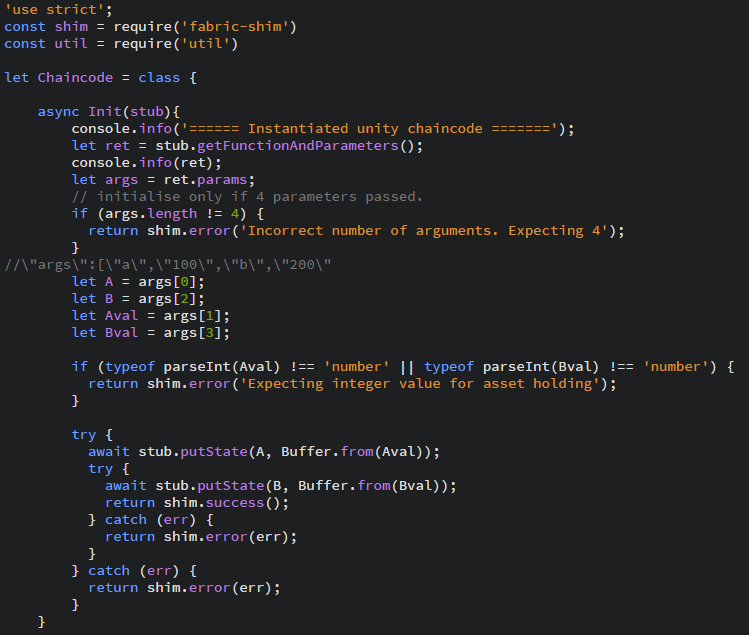
Figure: supply chain web homepage

* + 1. Program Architecture

The figure about material supply chain program architecture is below. At first start Fabric and deploy chaincde. Then program REST API to communicate with Fabric. The REST API is programed by ourselves, not by Fabric. This is safer that make REST API between Client and Fabric. And program Client Application to communicate to REST API. The Client Application here is presented by website.

* + 1. Program implementation

I choose javascript to develop Chaincode. There are two main functions, Init and Invoke. When instantiate Chaincode, it will call Init function first. While call Chaincode it will go through Invoke and then call other functions.

Figure: Init function

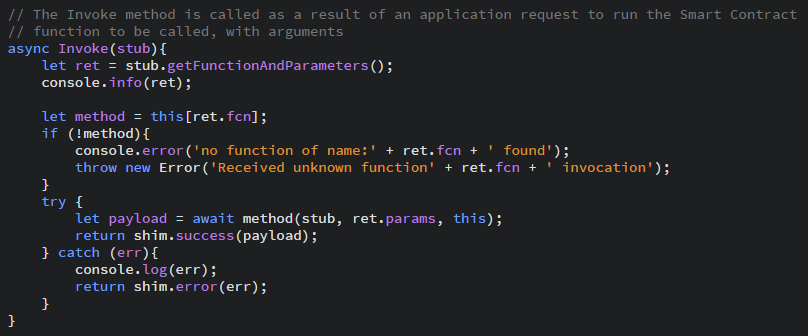


Figure: Invoke function

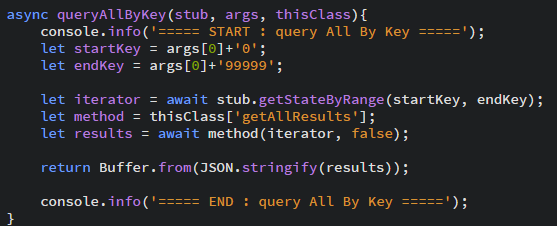






Figure: Partial function of the Chaincode

I program Client Web Application with javascript and communicate with Chaincode through REST API.

Figure: Client call REST API to communicate with chaincode

* 1. Tuna supply chain with Hyperledger Sawtooth(code source: <https://github.com/hyperledger/education/tree/master/LFS171x/sawtooth-material/sawtooth-tuna>)
     1. Introduction

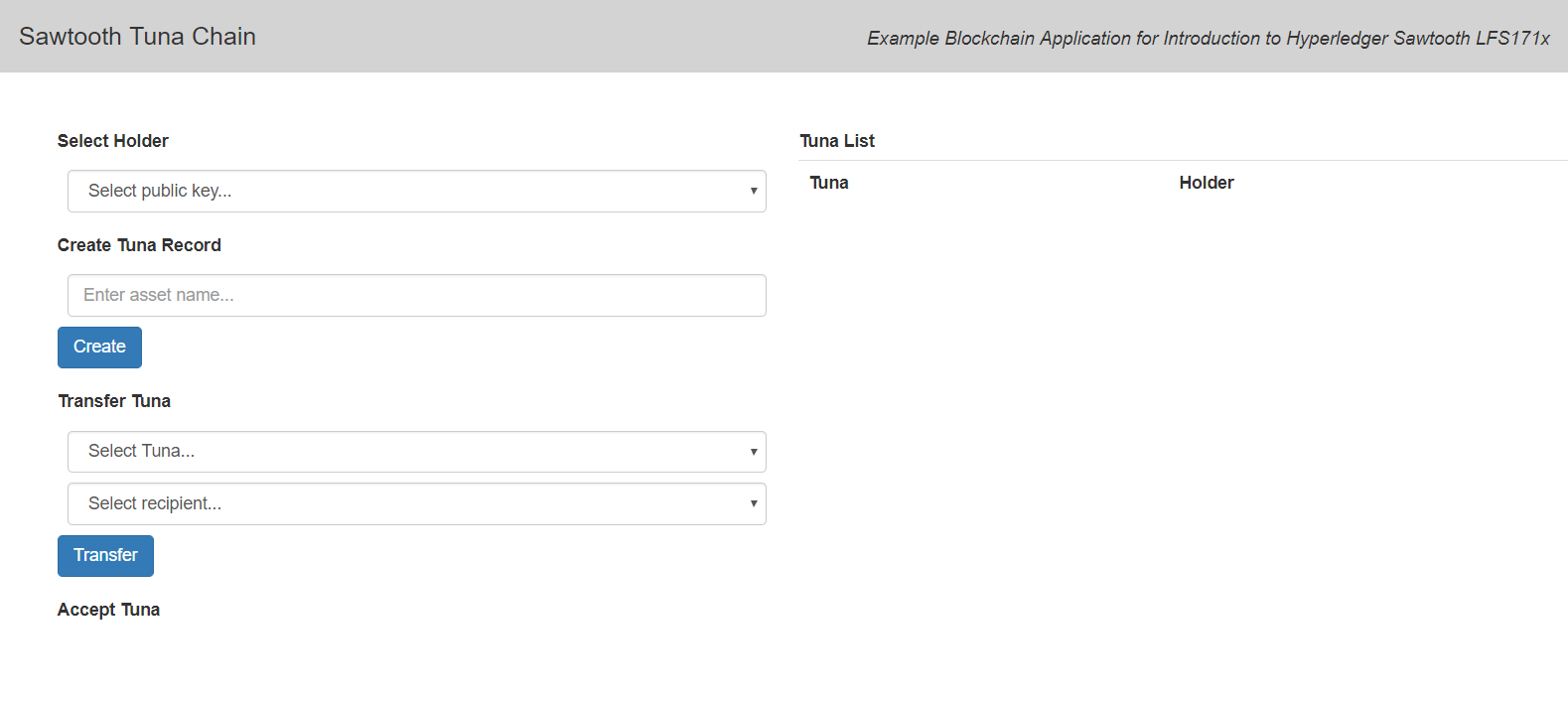
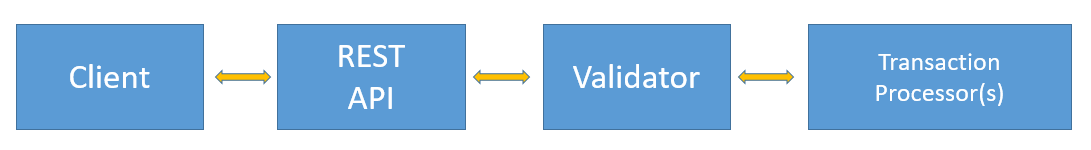
We record tuna data from fisher, wholesaler, transportation to restaurant, including where to catch it, species and temperature… and so on. Let the participants in supply chain know that tuna is safe and qualified.

Figure: Tuna home page

* + 1. Program Architecture

The figure about Hyperledger Sawtooth Tunachain program architecture is below. At first, start the Validator with docker, then Transaction Processor will be validated by Validator. The REST API is provided by Sawtooth. Client Applications have to communicate with Validator through REST API, then Validator validate the Transaction Batch is safe. If the Batch is safe, Validator will call Transaction Processor to execute server side logic.

* + 1. Program implementation

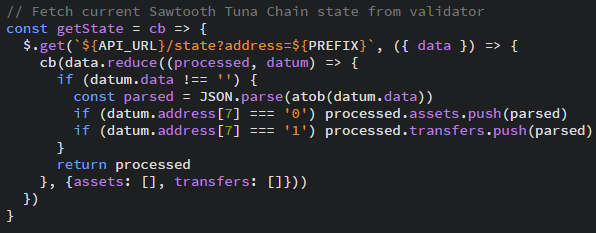
Processor is programmed with python and Client Application is programmed by javascript in Tunachain. Set data state in processor and

Program server side function. Program apply function to call other functions in handler. Then start the transaction processor and to be validated by validator.

Figure: Setting data state

Figure: Setting function logic

Figure: Define apply function to call different function in handler

Figure: Client call REST API to communicate with validator

* 1. The comparison in development supply chain application between Fabric and Sawtooth
     1. Smart Contract

REST API

About REST API communicating with smart contract, there is REST API for Sawtooth, but there is not REST API for Fabric.

Smart Contract Language

The support language for smart contract is different. There are more language support for Sawtooth, including Go, Javascript, Python, C++, Java and Rust. There are Go, Nodejs and Java for Fabric Chaincode.

Similar function

There is a similar concept in both smart contract. They both have to call other functions through one function. In Chaincode is invoke. In Processor is apply.

* + 1. Client

Code

The architecture about client side code is very different. The code of Fabric Client side include CA enrollment, key setting, Channel connection, request setting and receiving proposal response … and so on. The code of Sawtooth Client side include key setting, transaction setting, batch setting and communicating with Validator through REST API … and so on. And in Sawtooth, namespaces define the addressing scheme for application data. All Sawtooth applications store data in the state dictionary at 35-byte (70 hex-character) addresses.

Client Side Language

The support language for Client Side is different. There are more language support for Sawtooth, including Go, Javascript, Python and Rust. There are Nodejs and Java for Fabric now.

1. Real Applications
   1. Real Applications on Fabric
      1. Blockchain Cloud Platform

IBM publish blockchain cloud platform With Hyperledger Fabric. In the past two years, there are more than 400 enterprise blockchain projects and more than 250 trade on the platform.

* + 1. Food Security

IBM, Walmart and Tsinghua University have tried to use blockchain to trace the process of food supply chain. Including pork supply chain in China and mango supply chain in America. Walmart’s experiment shows that the process of tracing a pack of mango decrease from a few days to only two seconds.

* + 1. World Trade Supply Chain

IBM and Maersk build the world trade digital platform to raise the efficiency and transparency during transportation. Hyperledger Fabric Blockchain is the foundation of the platform. To increase efficiency and transparency between departments with immutability and distributed ledger.

* + 1. Financial

On 10th, January, 2017, IBM and Postal Savings Bank of China published asset keeping system with Hyperledger Fabric Blockchain. Implement sharing information, decrease redundant checking process and implement asset keeping supervision with distributed ledger, Smart Contract, privacy protection and Consensus… and so on. Decrease about 60 to 80 percent of original procedure with Blockchain.

* + 1. There are more real applications on Fabric:

https://medium.com/coinmonks/hyperledger-projects-in-real-companies-35016745362c

Financial Institutions: CLS

Carbon emission reduction:

In use by the carbon asset market in China, it allows enterprises to generate carbon assets more efficiently, helping to build a green, low-carbon and environmentally-friendly future in China. Carbon asset development, also known as CER (Carbon Emission Reduction) quota issuing, is one popular ways of encouraging enterprises to decrease emissions and use low carbon emission technology.

Healthcare:

IBM Watson Health (NYSE: IBM) has signed a research initiative with the U.S. Food and Drug Administration (FDA) aimed at defining a secure, efficient and scalable exchange of health data using blockchain technology.

Food Safety

Banks

……

* 1. Real Applications on Sawtooth
     1. Sawtooth Supply Chain(example、demo) https://github.com/hyperledger/sawtooth-supply-chain

https://sawtooth.hyperledger.org/examples/seafood.html

Sawtooth supply chain is an application to help you track the source and other contextual information of any asset. You can use it as is or customize it for different cases of use.

* + 1. Sawtooth Marketplace(example、demo) https://github.com/hyperledger/sawtooth-marketplace

https://sawtooth.hyperledger.org/examples/marketplace.html

Sawtooth Marketplace is an application that allows users to exchange customized amounts of “Assets” with other users in the blockchain.

* + 1. Sawtooth Private UTXO(example) https://github.com/hyperledger/sawtooth-private-utxo

Demonstrates how assets can be created and marketed. It shows how to use SGX to allow assets to be transferred to the general ledger and traded privately, where only the parties involved know the details of the transaction.

* + 1. Bond - Asset Settlement(Demo)

https://sawtooth.hyperledger.org/examples/bond.html

To streamline the process of transferring bonds, Sawtooth created a user interface and transaction family that allows investors to track and transfer bonds. Here, users can create, buy, sell, and settle their portfolio of bonds. While a transaction family was customized specifically for bonds and their unique identifiers, transaction families can be customized for a wide range of financial investment tools.

1. Resource
   * 1. <https://sawtooth.hyperledger.org/docs/core/releases/1.0/introduction.html>
     2. Edx course: LinuxFoundationX: LFS171x Blockchain for Business - An Introduction to Hyperledger Technologies
     3. Sawtooth 1.0 powerpoint
     4. Know Hyperledger Fabric? Then moving to Sawtooth is easy <https://hackernoon.com/know-hyperledger-fabric-then-moving-to-sawtooth-is-easy-15445f902493>
     5. Hyperledger Sawtooth Vs Hyperledger Fabric <https://www.oodlestechnologies.com/blogs/Hyperledger-Sawtooth-Vs-Hyperledger-Fabric>
     6. Hyperledger fabric samples balance transfer <https://github.com/hyperledger/fabric-samples/tree/release-1.3/balance-transfer>
     7. 商業場景讓區塊鏈走向現實 <https://new.qq.com/omn/20180413/20180413B191RL.html>
     8. Hyperledger Sawtooth examples <https://sawtooth.hyperledger.org/examples/>
     9. A processor at the heart of Hyperledger Sawtooth <https://medium.com/coinmonks/a-processor-at-the-heart-of-hyperledger-sawtooth-eng-763900f204f2>
     10. Hyperledger projects in real companies <https://medium.com/coinmonks/hyperledger-projects-in-real-companies-35016745362c>