Hyperledger Fabric version 1.0

By the end of this chapter, you will be able to set up a sample Hyperledger Fabric network, along with a simple application using the Fabric Node.JS SDK.

Chaincode (Hyperledger Fabric smart contract)

Hyperledger Fabric was the first proposal to The Linux Foundation's Hyperledger Project, contributed in part by IBM.

It's specifically formulated for industries such as

* manufacturing,
* agriculture,
* healthcare, and
* capital markets.

About the Demonstrated Scenario

"Illegal, unreported, and unregulated (IUU) fishing represents a theft of around 26 million tonnes, or close to $24 billion value of seafood a year."

As much as 40% of our oceans are heavily affected by human activities like illegal fishing.

our goal is to eliminate illegal, unreported, and unregulated fishing.

We will use Hyperledger Fabric to bring transparency and clarity to a real-world example: the supply chain of tuna fishing.

1. We will be describing how tuna fishing can be improved, starting from the source, fisherman Sarah, and the process by which her tuna ends up at Miriam's restaurant.
2. In between, we'll have other parties involved, such as the regulator who verify the validity of the data and the sustainability of the tuna catches.
3. We will be using Hyperledger Fabric's framework to keep track of each part of this process.

Using private channels, regulators and restaurateurs can confirm whether a particular shipment of tuna was sustainably and legally sourced, without needing to see the details of the entire journey.

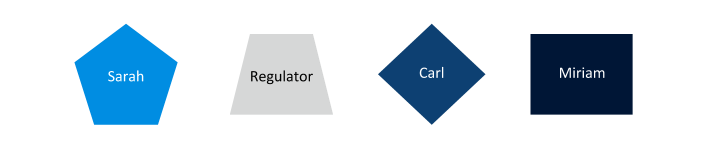
Defining Our Actors

**Sarah** is the fisherman who sustainably and legally catches tuna.

**Regulators** verify that the tuna has been legally and sustainably caught.

**Miriam** is a restaurant owner who will serve as the end user, in this situation.

**Carl** is another restaurant owner fisherman Sarah can sell tuna to.



Using Hyperledger Fabric, we will be demonstrating how tuna fishing can be improved starting from the source, fisherman Sarah, and the process by which she sells her tuna to Miriam's restaurant.

Featured Hyperledger Fabric Elements

**Channels** are data partitioning mechanisms that allow transaction visibility for stakeholders only. Each channel is an independent chain of transaction blocks containing only transactions for that particular channel.

The**chaincode** (Smart Contracts) encapsulates both the asset definitions and the business logic (or transactions) for modifying those assets. Transaction invocations result in changes to the ledger.

The**ledger** contains the current world state of the network and a chain of transaction invocations. A shared, permissioned ledger is an append-only system of records and serves as a single source of truth.

The**network** is the collection of data processing peers that form a blockchain network. The network is responsible for maintaining a consistently replicated ledger.

The**ordering service** is a collection of nodes that orders transactions into a block.

The**world state** reflects the current data about all the assets in the network. This data is stored in a database for efficient access. Current supported databases are LevelDB and CouchDB.

The**membership service provider** (MSP) manages identity and permissioned access for clients and peers.

The Catch

Through a client application, Sarah is able to gain entry to a Hyperledger Fabric blockchain network comprised of other fishermen, as well as regulators and restaurant owners.

Sarah has the ability to add to and update information in the blockchain network's  ledger as tuna pass through the supply chain, while regulators and restaurants have read access to the ledger.

After each catch, Sarah records information about each individual tuna, including:

1. a unique ID number,
2. the location and time of the catch,
3. its weight,
4. the vessel type, and
5. who caught the fish.

These details are saved in the world state as a key/value pair based on the specifications of a chaincode contract, allowing Sarah’s application to effectively create a transaction on the ledger.

You can see the example below:

**$ var tuna = { id: ‘0001’, holder: ‘Sarah’, location: { latitude: '41.40238', longitude: '2.170328'}, when: '20170630123546', weight: ‘58lbs’, vessel : ‘9548E’ }**

The Incentives

Miriam is a restaurant owner looking to source low cost, yet high quality tuna that have been responsibly caught.

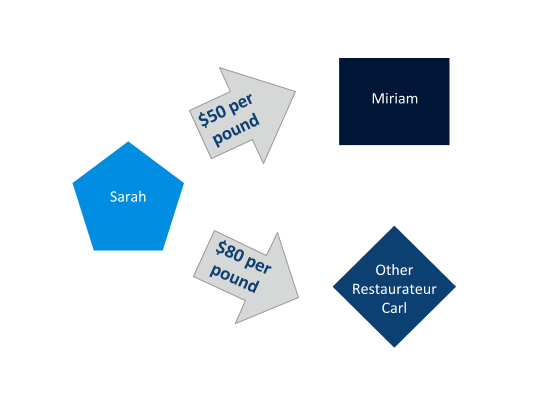
Sarah strives to make a living selling her tuna at a reasonable price. She would also like autonomy over who she sells to and at what price.

Luckily for both Sarah and Miriam, Hyperledger Fabric can help!

The Sale

Normally, Sarah sells her tuna to restaurateurs, such as Carl, for $80 per pound. However, Sarah agrees to give Miriam a special price of $50 per pound of tuna, rather than her usual rate.

In a traditional public blockchain, once Sarah and Miriam have completed their transaction, the entire network is able to view the details of this agreement, especially the fact that Sarah gave Miriam a special price. As you can imagine, having other restaurateurs, such as Carl, aware of this deal is not economically advantageous for Sarah.



To remedy this, Sarah wants the specifics of her deal to not be available to everyone on the network, but still have every actor in the network be able to view the details of the fish she is selling. Using Hyperledger Fabric's feature of **channels**, Sarah can privately agree on the terms with Miriam, such that only the two of them can see them, without anyone else knowing the specifics.

The Regulators

Regulators will also gain entry to this Hyperledger Fabric blockchain network to confirm, verify, and view details from the ledger.

Their application will allow these actors to query the ledger and see the details of each of Sarah’s catches to confirm that she is legally catching her fish.

Regulators only need to have query access, and do not need to add entries to the ledger.

they may be able to adjust who can gain entry to the network and/or be able to remove fishermen from the network, if found to be partaking in illegal activities.

Gaining Network Membership

Hyperledger Fabric is a permissioned network, meaning that only participants who have been approved can gain entry to the network. To handle network membership and identity, **membership service providers** (MSP) manage user IDs, and authenticate all the participants in the network.

A Hyperledger Fabric blockchain network can be governed by one or more MSPs

In our scenario only below persons should be the allowed to join the network.

* the regulator,
* the approved fishermen, and
* the approved restaurateurs

In configuring this MSP, certificates and membership identities are created. Policies are then defined to dictate the read/write policies of a channel, or the endorsement policies of a chaincode.

Our scenario has two separate chaincodes, which are run on three separate channels.

The two chaincodes are:

1. one for the price agreement between the fisherman and the restaurateur, and
2. one for the transfer of tuna.

The three channels are:

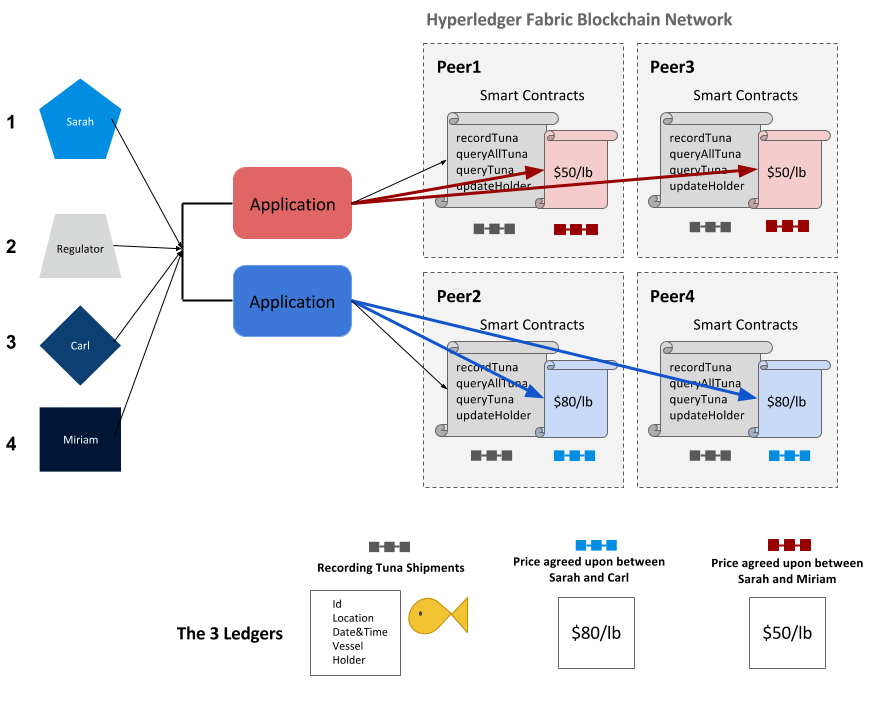
1. one for the price agreement between Sarah and Miriam;
2. one for the price agreement between Sarah and Carl; and
3. one for the transfer of tuna.

Each member of this network knows about each other and their identity. The channels provide privacy and confidentiality of transactions.

In Hyperledger Fabric, MSPs also allow for dynamic membership to add or remove members to maintain integrity and operation of the supply chain. For example, if Sarah was found to be catching her fish illegally, she can have her membership revoked, without compromising the rest of the network. This feature is critical, especially for enterprise applications, where business relationships change over time.

Summary of Demonstrated Scenario

1. Sarah catches a tuna and uses the supply chain application’s user interface to record all the details about the catch to the ledger. Before it reaches the ledger, the transaction is passed to the endorsing peers on the network, where it is then endorsed. The endorsed transaction is sent to the ordering service, to be ordered into a block. This block is then sent to the committing peers in the network, where it is committed after being validated.
2. As the tuna is passed along the supply chain, regulators may use their own application to query the ledger for details about specific catches (excluding price, since they do not have access to the price-related chaincode).
3. Sarah may enter into an agreement with a restaurateur Carl, and agree on a price of $80 per pound. They use the blue channel for the chaincode contract stipulating $80/lb. The blue channel's ledger is updated with a block containing this transaction.
4. In a separate business agreement, Sarah and Miriam agree on a special price of $50 per pound. They use the red channel's chaincode contract stipulating $50/lb. The red channel's ledger is updated with a block containing this transaction.



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:

* **Clients**   
  Clients are applications that act on behalf of a person to propose transactions on the network.

**Peers**  
Peers maintain the state of the network and a copy of the ledger. There are two different types of peers: **endorsing** and **committing**

- Endorsers simulate and endorse transactions  
- Committers verify endorsements and validate transaction results, prior to committing transactions to the blockchain.

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

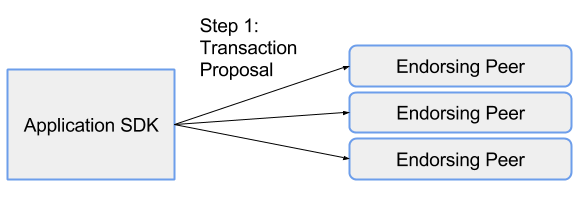
How to Reach Consensus

In a distributed ledger system, **consensus** is the process of reaching agreement on the next set of transactions to be added to the ledger. In Hyperledger Fabric, consensus is made up of three distinct steps:

* Transaction endorsement
* Ordering
* Validation and commitment.

Transaction Flow (Step 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.

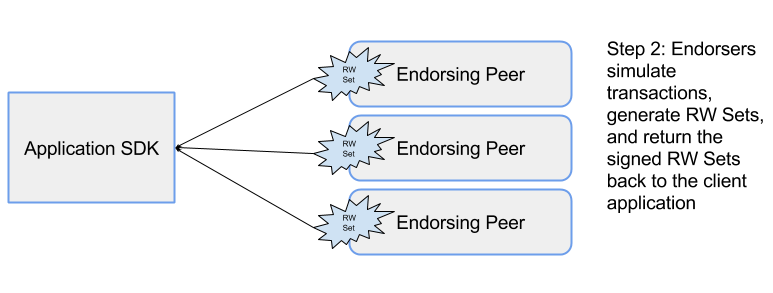
Application developers can leverage the Hyperledger Fabric network through the application SDK.

Transaction Flow (Step 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 capture what was read from the current world state while simulating the transaction, as well as what would have been written to the world state had the transaction been executed.

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

A transaction endorsement is a signed response to the results of the simulated transaction.

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.

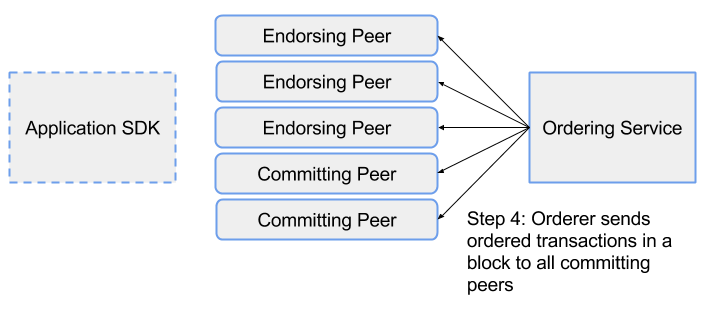
Transaction Flow (Step 3)

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 (Step 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 Fabric v1.0 architecture has been designed such that the specific implementation of 'ordering' (Solo, Kafka, BFT) becomes a pluggable component. The default ordering service for Hyperledger Fabric is Kafka. Therefore, the ordering service is a modular component of Hyperledger Fabric.

Ordering Service (Chris Ferris)

Ordering (Part I)

In a blockchain network, transactions have to be written to the shared ledger in a consistent order. The order of transactions has to be established to ensure that the updates to the world state are valid when they are committed to the network.

Unlike the Bitcoin blockchain, where ordering occurs through the solving of a cryptographic puzzle, or mining, Hyperledger Fabric allows the organizations running the network to choose the ordering mechanism that best suits that network.

This modularity and flexibility makes Hyperledger Fabric incredibly advantageous for enterprise applications.

Ordering (Part II)

Hyperledger Fabric provides three ordering mechanisms: SOLO, Kafka, and Simplified Byzantine Fault Tolerance (SBFT)

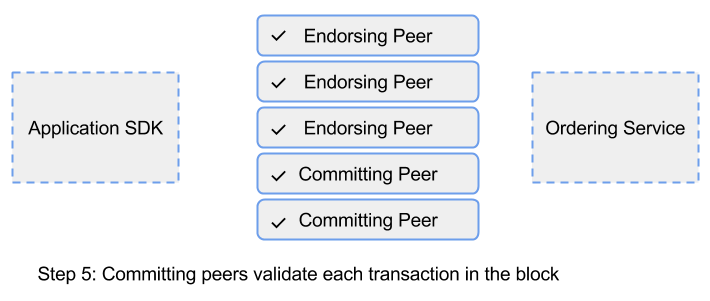
* **Kafka** is the Hyperledger Fabric ordering mechanism that is recommended for production use. This ordering mechanism utilizes Apache Kafka, an open source stream processing platform that provides a unified, high-throughput, low-latency platform for handling real-time data feeds. In this case, the data consists of endorsed transactions and RW sets. The Kafka mechanism provides a crash fault-tolerant solution to ordering.

Transaction Flow (Step 5)

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

Specifically, that the Read data that existed when the endorsers simulated the transaction is identical to 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.

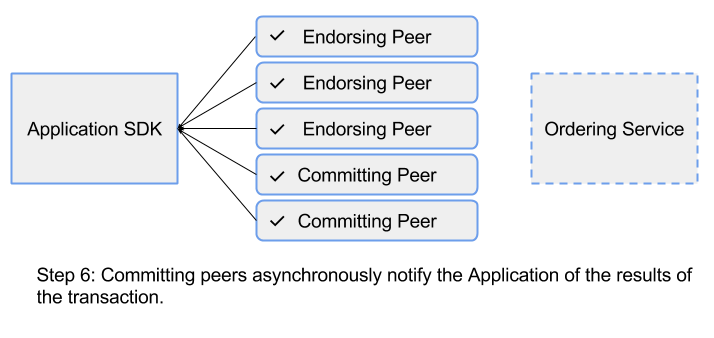


If the transaction fails, that is, if the committing peer finds that the RW set does not match the current world state, the transaction ordered into a block will still be included in that block, but it will be marked as invalid, and the world state will not be updated.

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 (Step 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.

Transaction Flow Summary

It is important to note that the state of the network is maintained by peers, and not by the ordering service or the client.

Normally, you will design your system such that different machines in the network play different roles. That is, machines that are part of the ordering service should not be set up to also endorse or commit transactions, and vice versa.

However, there is an overlap between endorsing and committing peers on the system.

*Steps -:*

* Endorsing peers must have access to and hold smart contracts, in addition to fulfilling the role of a committing peer.
* Endorsing peers do commit blocks, but committing peers do not endorse transactions.
* Endorsing peers verify the client signature, and execute a chaincode function to simulate the transaction. The output is the chaincode results, a set of key/value versions that were read in the chaincode (Read set), and the set of keys/values that were written by the chaincode.
* The proposal response gets sent back to the client, along with an endorsement signature.
* These proposal responses are sent to the orderer to be ordered.
* The orderer then orders the transactions into a block, which it forwards to the endorsing and committing peers.
* The RW sets are used to verify that the transactions are still valid before the content of the ledger and world state is updated.
* Finally, the peers asynchronously notify the client application of the success or failure of the transaction.

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.

Installing Hyperledger Fabric Docker Images and Binaries

Installing Hyperledger Fabric

Starting a Test Hyperledger Fabric Network

We’ll create a simple two member network consisting of two organizations (effectively, Sarah and Miriam), each maintaining two peers and an ordering service.

We will use Docker images to bootstrap our first Hyperledger Fabric network. It will also launch a container to run a scripted execution that will join peers to a channel, deploy, and instantiate the chaincode, and execute transactions against the chaincode.

## Chaincode

* In Hyperledger Fabric, **chaincode** is the 'smart contract' that runs on the peers and creates transactions
* It enables users to create transactions in the Hyperledger Fabric network's shared ledger and update the world state of the assets.
* Chaincode is written in Go.
* The chaincode manages the ledger state.
* Assets are created and updated by a specific chaincode, and cannot be accessed by another chaincode.
* Applications interact with the blockchain ledger through the chaincode. Therefore, the chaincode needs to be installed on every peer that will endorse a transaction and instantiated on the channel.

There are two ways to develop smart contracts with Hyperledger Fabric:

* Code individual contracts into standalone instances of chaincode.
* (More efficient way) Use chaincode to create decentralized applications that manage the lifecycle of one or multiple types of business contracts, and let the end users instantiate instances of contracts within these applications.

Chaincode Key APIs

An important interface that you can use when writing your chaincode is

1. [ChaincodeStub](https://godoc.org/github.com/hyperledger/fabric/core/chaincode/shim" \l "Chaincode" \t "_blank)
2. [ChaincodeStubInterface](https://godoc.org/github.com/hyperledger/fabric/core/chaincode/shim#ChaincodeStub)

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

The key APIs for chaincode include:

When creating a chaincode, there are two methods that you will need to implement:

* **Init**  
  Called when a chaincode receives an instantiate or upgrade transaction. This is where you will initialize any application state.
* **Invoke**  
  Called when the ***invoke*** transaction is received to process any transaction proposals.

The chaincode must be installed using the

**peer chaincode install** command,

and instantiated using the

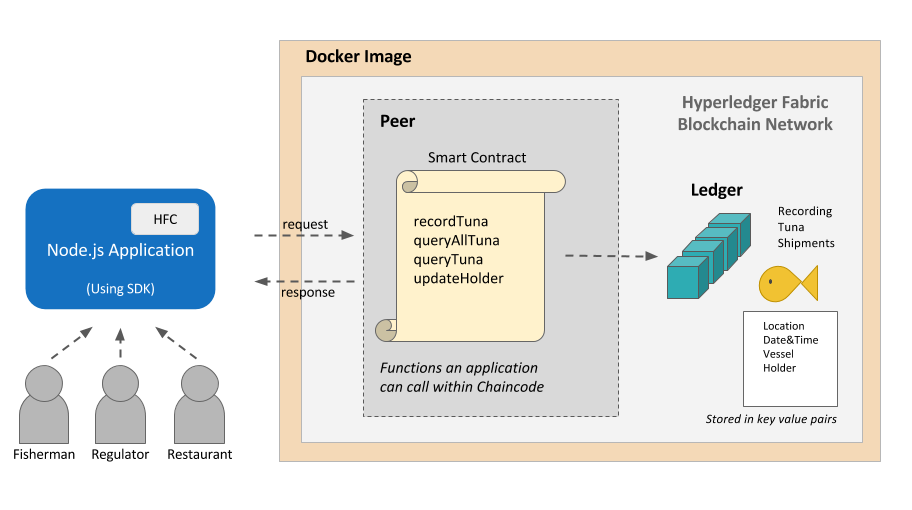
**peer chaincode instantiate** command

before the chaincode can be invoked. Then, transactions can be created using the

**peer chaincode invoke** or

**peer chaincode query** commands.

## 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.

## Hyperledger Sawtooth Application Flow

Hyperledger Sawtooth allows entities to securely update and read the distributed ledger without involving a central authority. Developers create application and transaction processor business logic (smart contract).

Through the client application, users (fisherman, regulator, restaurant) are able to modify the state by creating and applying transactions.

Through a REST API, the client application creates a batch containing a single transaction, and submits it to the validator.

The validator applies the transaction using the transaction processor, which makes a change to the state (e.g., creating a record of a tuna catch).

