4
Hyperledger – Blockchain for Businesses

After understanding the architecture, core components, and the process of blockchain technology, it is important to explore the possibilities in regard to fulfilling business needs. Blockchain is responsible for running distributed networks without third-party regulators. It is now becoming an essential component to consider and this will shape the next generation of financial technology and governance models. However, blockchains used for cryptocurrency are highly focused on rewards and giving incentives to their participants, such as Bitcoin and Ethereum. To overcome this challenge, the Linux Foundation and industry leaders have collaborated to form a distributed ledger-based project named the Hyperledger project. In this chapter, you will learn about how the Hyperledger project is different from existing blockchain technologies, its core components, transaction flow, and turning up an application with Hyperledger technology.

You will learn the following topics in this chapter:

- Hyperledger overview
- Blockchain as a service
- Architecture and core components
- Hyperledger Fabric model
- Bitcoin versus Etherum versus Hyperledger
- Hyperledger Fabric capabilities
- Lab using the Tuna application

Technical requirements

This chapter consists of a lab to demonstrate the Hyperledger application to solve real-life challenges in supply chain management. It is required that you have the source code from the following link: `https://github.com/hyperledger/education.git`.

Hyperledger overview

Hyperledger is an open source initiative focused on covering core industry needs with distributed ledger technologies. It's a group program hosted by the Linux Foundation in collaboration with several industry giants in information technology, banking, logistics, transportation, finance, manufacturing, and IoT.

Although cryptocurrency is still struggling to gain trust with several government and corporate bodies, blockchain is being adopted as a key to secure business operations and management technology. Because of the rigid and static nature of Bitcoin, it can't be used for business application purposes. Although Ethereum has the capability of turning up business applications with its smart contracts, because of its permissionless use cases, financial institutions and other critical business operations have hesitated to try the Ethereum blockchain.

Hyperledger is the only distributed ledger technology framework that was built to be granular for businesses that were in need of permissioned blockchains to achieve better control over an entire system. Hyperledger does not support any cryptocurrency platform or related system, as it is more about solving critical business problems.

Founded in December 2015, Hyperledger has been appreciated and adopted by several industry leaders such as Accenture, Airbus, American Express, Cisco, Fujitsu, Hitachi, IBM, Intel, SAP, NEC, BBVA, Bitmark, Bosch, CA Technologies, Capgemini, EY, Factom, H3C, NSE, Oracle, PwC, Redhat, Samsung, Ripple, Thales, Wipro, the Cloud Security Alliance, and many more.

The Hyperledger project was also planned for collaboration between every blockchain enthusiast, blockchain communities, corporates, and nonprofit organizations with a single and comprehensive standard of building distributed ledger applications. In the way that WordPress revolutionized the method and turn-up time for a website, Hyperledger is on its way to reducing the cost and overall time in turning up a distributed ledger application.

Blockchain-as-a-service (BaaS)

Since the birth of cloud computing, one of the hottest terms that has changed the way a product or a service can be delivered or deployed is *X (anything) as a service,* where *X* is any form of software or application. After the world recognized the immersive power of blockchain, industry leaders began to explore various possibilities of using blockchain with their existing cloud infrastructure models such as supply chain management, identity and access control, database management, and many more. Hyperledger resembles to the distributed ledger technology however blockchain technology has been taken a special focus in the ecosystem.

With the Azure Blockchain service, Microsoft became the first software vendor to launch BaaS in 2015. Microsoft, in close collaboration with ConsenSys, announced that it was going to develop an Ethereum BaaS on the Microsoft Azure platform. SAP launched its own BaaS platform and named it *Leonardo,* which is a Hyperledger-based cloud service.

Deloitte, the largest consulting firm, has come up with a blockchain-based business solution and named it Rubix Core. It is an architecture designed for building private and customized networks for their clients.

Program goal

The Hyperledger project has been widely appreciated for its upfront effort to develop cross-industry frameworks for platform collaboration. The financial industry has been the most active in collaborating with Hyperledger platforms to achieve a seamless move. Let's understand the goals of the Hyperledger project to get a sense of its roadmap:

- **Community-driven infrastructure**: As the Hyperledger project is supported by several private and government institutions, it presents a highly efficient and open community-driven environment
- **Enterprise-grade framework**: Unlike the cryptocurrency blockchain, Hyperledger was developed to support businesses to perform secure and reliable transactions over distributed ledger networks
- **Building technical communities**: The project is also aimed at building a more effective and larger technical community to innovate and develop blockchain smart contacts and other related code
- **Awareness**: It's a great way to spread awareness to businesses and other institutions about blockchain technology and its business use cases

Architecture and core components

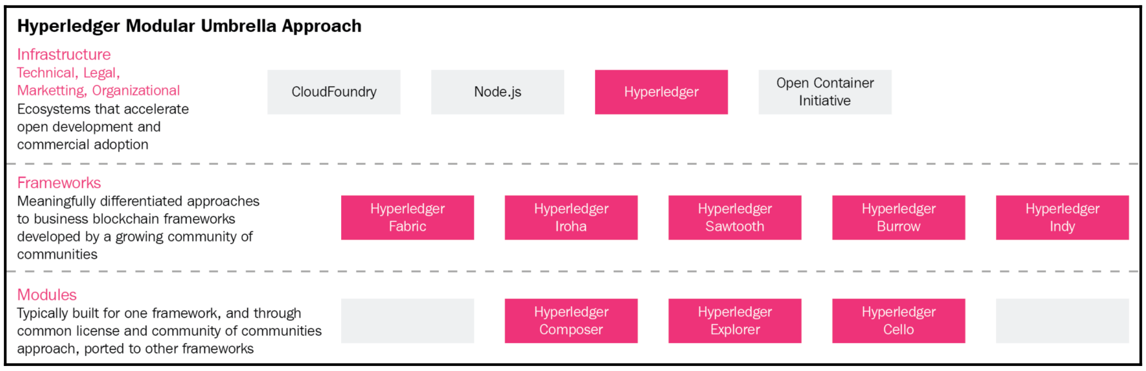
Hyperledger is an open source framework that allows businesses to build enterprise-grade solutions based on distributed ledger technology. This framework consists of the following building blocks:

- **Shared ledger**: It is an append-only ledger, and it stores the blocks in chronological order
- **Consensus algorithm**: It's a method to achieve a common agreement over a change in the distributed ledger
- **Privacy**: The main purpose of building the Hyperledger was to achieve a permissioned network for secure and reliable transactions in mission-critical business environments
- **Smart contract**: This is a granular method to plan and process transaction requests

Let's understand the Hyperledger architecture:

- **Consensus layer**: This is mainly responsible for generating an agreement on each order and validating transactions based on a predefined set of rules
- **Smart contract layer**: This takes care of transaction requests and applying business logic
- **Communication layer**: This facilitates a platform that allows nodes to communicate over peer-to-peer transport
- **Data store abstraction**: This allows various data sources to be used by other modules
- **Crypto abstraction**: This allows different crypto algorithms to be used without impacting other modules
- **Identity service**: This enables the deployment of root of trust during blockchain setup with additional authentication and authorization cover
- **Policy service**: This is responsible for managing several policies, such as the consensus policy, endorsement policy, and the group management policy
- **API**: This enables clients and applications to talk to blockchain modules
- **Interoperation**: This provides interoperability among different blockchain instances

Let's understand various Hyperledger frameworks, which are as follows:


Hyperledger Modular Umbrella Approach
Infrastructure
Technical, Legal, Marketting, Organizational
Ecosystems that accelerate open development and commercial adoption
CloudFoundry
Node.js
Hyperledger
Open Container Initiative
Frameworks
Meaningfully differentiated approaches to business blockchain frameworks developed by a growing community of communities
Hyperledger Fabric
Hyperledger Iroha
Hyperledger Sawtooth
Hyperledger Burrow
Hyperledger Indy
Modules
Typically built for one framework, and through common license and community of communities approach, ported to other frameworks
Hyperledger Composer
Hyperledger Explorer
Hyperledger Cello

- **Iroha**: Hyperledger Iroha is a blockchain framework contributed by Soramitsu, Hitachi, NTT DATA, and Colu. It is designed to be used by mobile application developers under Android and iOS packages. It has a simple design with the C++ programming package along with the YAC consensus algorithm.
- **Sawtooth**: It is contributed to by Intel and is built to use several consensus algorithms based on the size of the network. By default, Hyperledger Sawtooth uses **Proof of Elapsed Time** (**PoET**) to achieve consensus among nodes. It is designed for versatility to support both permissioned and permissionless implementations.
- **Indy**: Hyperledger Indy is a distributed ledger to achieve business solutions for decentralized identities and provides interoperability across several supporting **Distributed Ledger Technologies** (**DLTs**). It is designed to achieve privacy across nodes and throughout transactions.
- **Burrow**: Hyperledger Burrow is a permissionable smart contract that provides a modular blockchain client with a permissioned smart contract interpreter built with **Ethereum Virtual Machine** (**EVM**).

Hyperledger Fabric model

The Hyperledger Fabric project is powered by the IBM blockchain platform and is hosted with the Linux Foundation, with its key highlight over confidential transactions being a permissioned network, programmable business login, and no need for cryptocurrency computational methods.

"We're seeing great results and actively preparing for the transition to 1.1.0. Our latest offering, the IBM Blockchain Platform Starter Plan, will be among one of the first products in the market to deliver on this new release"

—Jerry Cuomo, VP of IBM blockchain technology.

After getting an insight into Hyperledger Fabric and other projects under the Hyperledger project umbrella, it's time to understand the technology practically, and also see some of the challenges during its deployment steps.

Hyperledeger Fabric core components

After understanding the transaction flow, it is important to get insight into how communication is established and maintained among several nodes of the network:

- **Nodes:** There are the following three roles in the Hyperledger network:
    - **Clients**: Clients propose the transaction request on the network. It has to be connected to a peer to participate in blockchain. The client has the right to connect the desired peer to the network.
    - **Peers**: Peers listen to the ledger update and keep a copy of it. Based on their nature, there could be two further types:
        - **Endorsing peers**: Endorsers simulate and endorse transactions
        - **Committing peers**: Committers validate transactions before committing transactions in the network
    - **Ordering service**: The ordering service accepts endorsed transactions, arranges and orders them into a block, and finally delivers it to committing peers. The ordering service also provides a shared and secure communication channel for clients and peers. It acts as a medium for broadcasting transactions and helps us deliver this to peers.
- **Ledger**: Just like with Bitcoin and Ethereum, the Hyperledger ledger provides a verified list of all valid and invalid transactions throughout the system's operation. It is created by an ordering service and is kept with all the peers in the network.

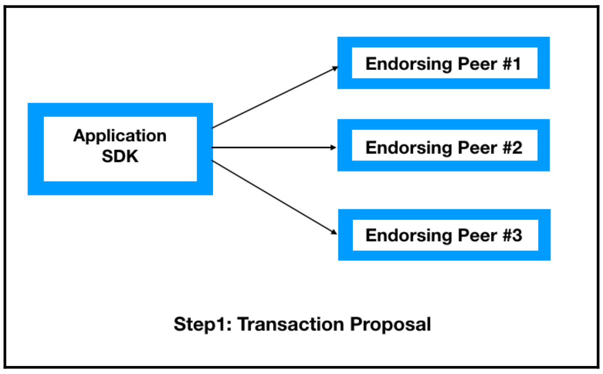
- **Channel**: The Hyperledger Fabric channel is a restricted communication medium for nodes to conduct confidential transactions. A channel is specific for a member, a shared ledger, a chaincode application, and the ordering service node. Each peer who joins the channel has to get a grant from the **Membership Service Provider** (**MSP**), which verifies each peer to its respective channel peers and services.
- **The world state:** This reflects the current state of data about all the assets in the network. The data is securely stored in the following format:
    - **LevelDB**: This is the default database for Hyperledger Fabric, and it simply stores key/value pairs.
    - **CouchDB**: This is best suited for web and native applications and it speaks JSON natively. It supports binary for all data storage needs.
- **Chaincode:** Chaincode manages the business logic agreed and created by members in the network. It is a program written in Go—Node.js:
    - **LevelDB**: This is a default programming language that runs over a secured Docker container and manages the ledger state.
    - **CouchDB**: This is another database programming language that stores JSON objects. It also supports key range, composite, and full data-rich queries.
- **Consensus:** Consensus is the process of achieving an agreement on a set of transactions to be added to the ledger. In Hyperledger Fabric, consensus is achieved with the following three steps:
    - Transaction endorsement
    - Ordering
    - Validation and commitment

Now, let's understand these consensus components and how they work with Hyperledger and its transaction processing methods.

Workings of Hyperledger and transaction processing

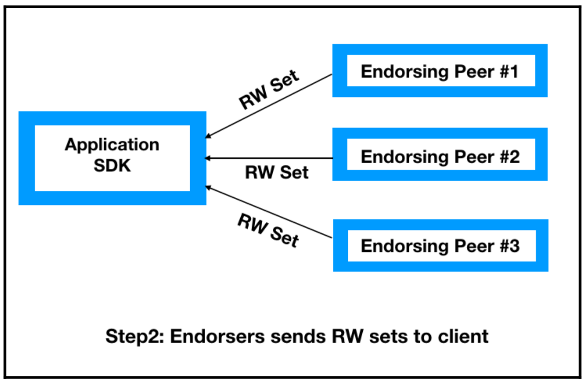
The workings of Hyperledger and transaction processing can be explained as follows:

1. **Transaction proposal**: In Hyperledger Fabric, the process starts with the client application sending transaction proposals:



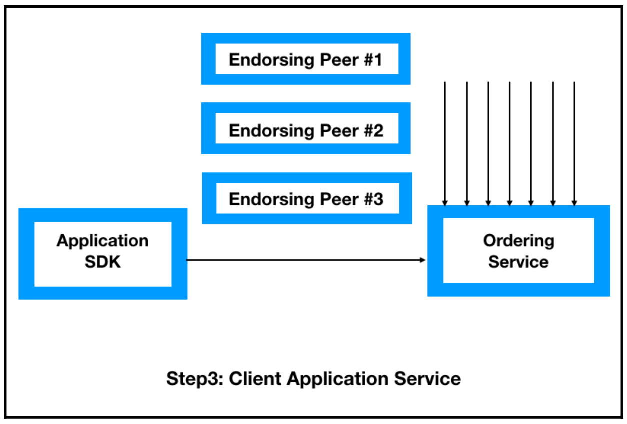
Each client application proposes transactions to endorse peers for the simulation and endorsement process.

2. **Endorsers send RW sets to the client**: Each endorsing peer simulates the proposed transaction and stores sets of read and written data named **RW sets**. These sets are signed by endorsing peers and are returned to the client application:

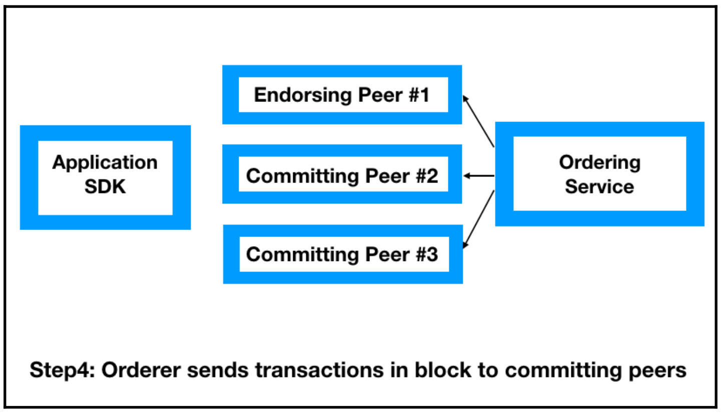


Transaction endorsement: This is a signed response, which results from the simulated transaction. There are several ways a transaction endorsement can be defined through policy chaincode, which is similar to a smart contract. One transaction endorsement policy resembles one defined chaincode.

3. **Client application service**: Once a client application receives the RW set and endorsed transactions, it has to submit these to the ordering service. This method keeps working, regardless of the transaction endorsement and RW set submitted by other client applications:



4. **An orderer sends transactions in blocks to committing peers**: The ordering service accepts both RW sets and endorsed transactions and arranges them into blocks, then forwards them to committing peers:

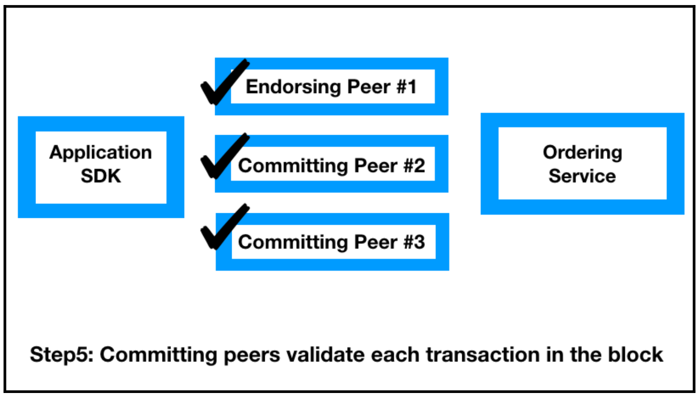


The ordering service is responsible for organizing all the transactions and then committing them to the ledger. By default, the ordering service for Hyperledger Fabric is Kafka, which is an open source, stream processing platform developed by the **Apache Software Foundation** (**ASF**).

Now, let's understand the workings of the ordering service in more detail. It is important to divide this into core parts:

- **Part 1 of the ordering service**: A block is created once a certain number of transactions is ready in a specified time frame and these transactions are committed in chronological order. Unlike the Bitcoin blockchain, Hyperledger Fabric provides the best suited ordering mechanism, and this helps organizations to design a granular, flexible, and scalable decentralized network.
- **Part 2 of the ordering service**: Hyperledger Fabric supports three ordering service mechanisms, SOLO, Kafka, and **Simplified Byzantine Fault Tolerance** (**SBFT**):
    - **SOLO**: This is best suited for software developers for research and testing purposes, and this has only one ordering node.
    - **Kafka**: This is another Hyperledger Fabric ordering mechanism that is production ready. It is developed by ASF and provides a unified, highly efficient, and low-latency software platform to handle real-time feeds. In Hyperledger Fabric, Kafka handles RW sets and endorsed transactions.
    - **SBFT**: This is similar to the PoW consensus mechanism of the Bitcoin blockchain. This solution is designed to overcome Byzantine failure, allowing the system to work even if there is a malicious node or a group of malicious nodes in the network.

5. **Committing peers validates each transaction in the block**: The committing peer validates the transactions to ensure the RW set matches the current world state. Once the committing peer validates the transaction, the transaction is updated to the ledger and the world state is automatically updated with write data from the RW set:



In the end, the committing peer has to notify the client application of the success or failure of the transaction.

6. **Identity verification**: At each step of the transaction flow, from endorsement to version check, identity verification remains a continuous process.

Bitcoin versus Ethereum versus Hyperledger

Blockchain is the most exciting innovation, and it is still popular in the cryptocurrency space. In the past few years, the industry has also recognized the impact of blockchain on their business model operation and management. Although blockchain technology works seamlessly in its native form, most of its business needs never fit the one-stop-shop solution. Hence, we have several versions of the blockchain network. Let's first understand their characteristics so that we can understand the comparison between some of the popular blockchain models in detail:

- **Permission restrictions**: This defines the eligibility of transaction processors to create or block the existing ledger. In this context, the following two models exist:
  - **Permissioned blockchain**: In this model, transactions processing can only be performed by preselected users. Hyperledger Fabric is an example of this.
  - **Permissionless blockchain**: This model doesn't restrict the transaction processor from creating or adding a new block. Ethereum and Bitcoin are some of the most popular examples of this.

- **Restricted access to data**: This specifies about the read rights across the blockchain network. There are the two following models:
    - **Public blockchain**: There is no restriction while reading the ongoing transactions. Anyone can download the updated blockchain ledger with the blockchain node client.
    - **Private blockchain**: In this model, access to the blockchain ledger is restricted to only preselected users.
- **The consensus mechanism**: In the distributed network, it is critical to achieve trustless networks and determine consensus for all transactions. This ensures that only valid and legitimate transactions are added to the blockchain. PoW, PoS, and PBFT are some examples of consensus algorithm.
- **Scalability**: Scalability depends on two factors, **nodes** and **performance**. Node scalability is the extent that nodes that can be added in the network without impacting the overall performance, and scalability depends on the number of transactions per seconds.
- **Anonymity**: This refers to the identity of users in the blockchain, which are made open or hidden.
- **Governance**: This is the level of decision making power distributed in the blockchain community. The blockchain platform has to be maintained by either the core developer team or other stakeholders.
- **Native currency**: This refers to the currency valid within the blockchain such as Bitcoin with the Bitcoin blockchain.
- **Scripting**: This refers to the level of programming supported by the **decentralized application** (**dApp**):

Characteristics	**Bitcoin**	**Ethereum**	**Hyperledger**
Permission restrictions	Permissionless	Permissionless	Permissioned
Restricted public access to data	Public	Public or private	Private
Consensus	PoW	PoW	PBFT
Scalability	High node scalability, low performance scalability	High node scalability, low performance scalability	Low node scalability, high performance scalability
Centralized regulation (governance)	Low, decentralized decision making by community/miners	Medium, core developer group, but EIP process	Low, open governance model based on the Linux model
Anonymity	Pseudonymity, no encryption of transaction data	Pseudonymity, no encryption of transaction of data	Pseudonymity, encryption of transaction data
Native currency	Yes, Bitcoin	Yes, ether	No

Scripting	Limited possibility of stack-based scripting	High possibility, tuning complete virtual machine, high-level language support—Solidarity	High possibility, tuning complete scripting of chaincode, high-level Go language

Hyperledger Fabric capabilities

Hyperledger comes with a full-stack, enterprise-grade business solution to deliver secure and scalable value with added security, confidentiality, and performance. Hyperledger Fabric delivers the following functionalities and core capabilities:

- **Identity management**: To turn a permissioned network, Hyperledger Fabric provides a membership identity service that maintains user IDs and then authenticates each of one of them in the network. One user ID can be allowed to invoke a chaincode application, but can be blocked to turn up a new chaincode.
- **Efficient processing**: Hyperledger assigns a role for each node based on transaction ordering and commitment. The overall performance improves as the concurrent execution increases and improves the time to deliver each order.
- **Privacy and confidentiality**: Private channels restrict the messaging paths to provide transaction privacy and confidentiality for specific network members. All data, including member information, transactions, and channel details, remains invisible and inaccessible to other network members.
- **Chaincode functionality:** This regards chaincode applications and is the business logic of Hyperledger Fabric. Chaincode ensures that all transactions that transfer ownership are subject to its rules and requirements. The operating parameters of the channel are usually defined by the system chaincode, whereas validation system chaincode defines the requirements for endorsing and validating transactions.

Lab

After understanding these insights into Hyperledger Fabric, with its architecture, components, transaction flow, and chaincode, it is now time to arrange each of these pieces in a lab. In order to keep the lab accessible, we are going to run a lab environment from GitHub hosted under `https://fabric-sdk-node.github.io/`.

Tuna application

The tuna application is about the transfer of tuna fish shipments between different parties in the supply chain. This entire application is written in Node.js, and gRPC is used to interact with the chaincode:

- **Aim**: Using the Fabric Node SDK, establish a connection with the Hyperledger blockchain. The peer will be configured to communicate to its application-specific chaincode container. By the end of this exercise, we will get familiar with how to use the Node.js SDK to communicate with the network. We will also gain an understanding of how an application chaincode network and ledger interact with one another.
- **Basic installation**: In case you haven't downloaded the `education` repository for this course, follow these directions in your Terminal window:

  ```
  $ git clone https://github.com/hyperledger/education.git
  $ cd education/LFS171x/fabric-material/tuna-app
  ```

  Make sure that you have Docker running on your machine before you run the next command. We need to make sure that we have completed the installation of the Hyperledger Fabric section in this chapter before moving on to this application section, as otherwise we will likely experience errors. First, remove any pre-existing containers, as they may conflict with commands in this tutorial:

  ```
  $ docker rm -f $(docker ps -aq)
  ```

  Now, let's start the Hyperledger Fabric network with the following command:

  ```
   $./startFabric.sh
  ```

- **Troubleshooting**: If after running the previous command you are getting an error similar to the following:

  ```
  ERROR: failed to register layer: rename
  /var/lib/docker/image/overlay2/layerdb/tmp/write-set-091347846
  /var/lib/docker/image/overlay2/layerdb/sha256/9d3227c1793b7494e
  598caafd0a5013900e17dcdf1d7bdd31d39c82be04fcf28: file exists
  ```

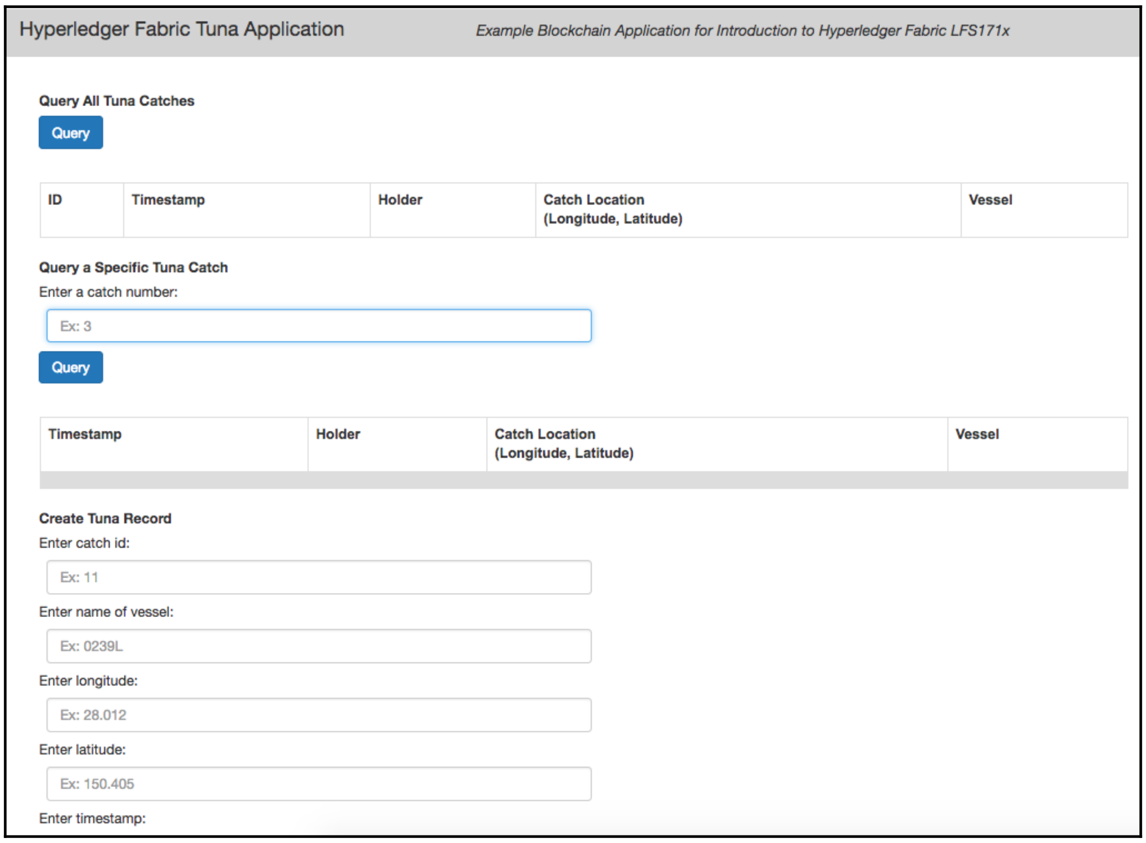
  Then, try running the following command:

  ```
  $ rm -rf
  ~/Library/Containers/com.docker.docker/Data/*
  ```

Install the required libraries from the `package.json` file, register the admin and user components of our network, and start the client application with the following commands:

```
$ npm install
$ node registerAdmin.js
$ node registerUser.js
$ node server.js
```

Load the client by simply opening `localhost:8000` in any browser window of your choice, and you should see the user interface for our simple application at this URL, as shown in the following screenshot:



ID	Timestamp	Holder	Catch Location (Longitude, Latitude)	Vessel

Timestamp	Holder	Catch Location (Longitude, Latitude)	Vessel

- **Troubleshooting**: If the client fails to connect to the Tuna server, we need to execute the following commands:

```
Error: [client-utils.js]: sendPeersProposal - Promise is
rejected: Error: Connect Failed error from query = { Error:
Connect Failed at /Desktop/prj/education/LFS171x/fabric-
material/tuna-
app/node_modules/grpc/src/node/src/client.js:554:15 code: 14,
metadata: Metadata { _internal_repr: {} } }
```

Try running the following commands:

```
 $ cd ~ $ rm -rf .hfc-key-store/.
```

Then, run the previous commands starting with:

```
$ node registerAdmin.js
// File Structure tuna-app/tuna-chaincode.go
```

This is the chaincode file that contains all our business logic for the sample Tuna app:

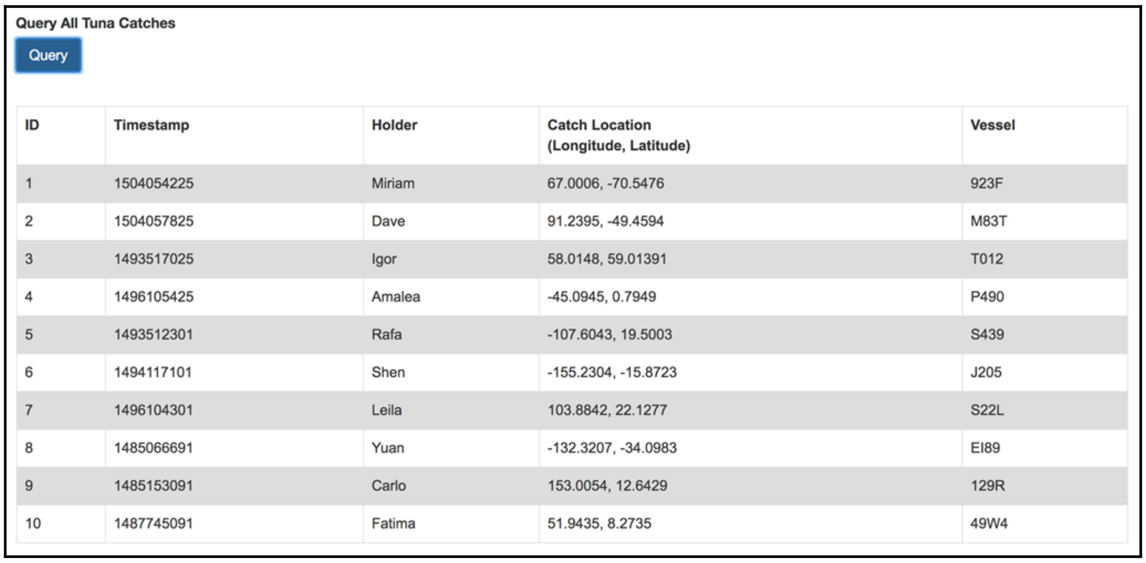
- `tuna-app/app.js`: This is JavaScript client code in `app.js` that manipulates HTML elements for the user interface.
- `tuna-app/index.html`: This is an HTML file containing the UI for the client application.
- `src`: This is a folder containing code that uses a **Software Development Kit** (**SDK**) to connect a client request to network and chaincode functions.
- `tuna-app/src/controller.js`: This contains functions that perform operations and interrogate data.
- `tuna-app/src/server.js`: `server.js` is used to view the UI at `localhost:8000`.

- **Verification**: Now, let's query our database, where there should be some sample entries; since our chaincode smart contract initiated the ledger with 10 previous catches, this function takes no arguments. As we can see on line 6, it takes an empty array:

```
// queryAllTuna - requires no arguments
const request = {
chaincodeId:'tuna-app',
txId: tx_id,
fcn: 'queryAllTuna',
args: ['']
};
return channel.queryByChaincode(request);
```

The code comes from `..src/queryAllTuna.js`.

Now, let's query our database, where there should be some sample entries already, since our chaincode smart contract initiated the ledger with the ten previous catches. This function takes no arguments, as we can see on line 6 in the preceding code. Instead, it takes an empty array. The query response that can be seen in the user interface is then pre-populated entries with the attributes for each catch:



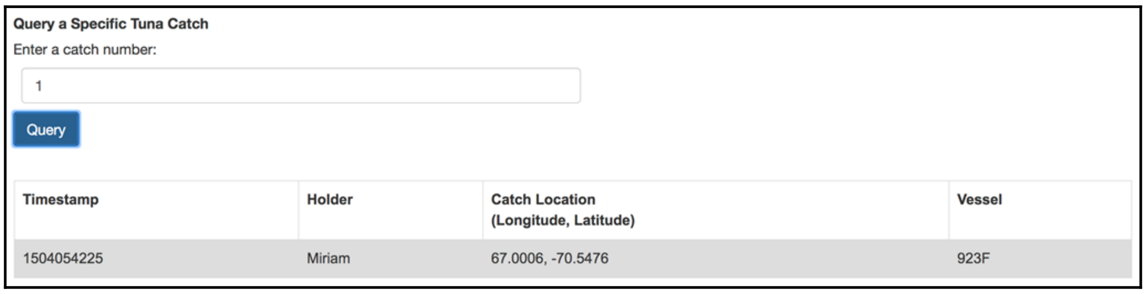
ID	Timestamp	Holder	Catch Location (Longitude, Latitude)	Vessel
1	1504054225	Miriam	67.0006, -70.5476	923F
2	1504057825	Dave	91.2395, -49.4594	M83T
3	1493517025	Igor	58.0148, 59.01391	T012
4	1496105425	Amalea	-45.0945, 0.7949	P490
5	1493512301	Rafa	-107.6043, 19.5003	S439
6	1494117101	Shen	-155.2304, -15.8723	J205
7	1496104301	Leila	103.8842, 22.1277	S22L
8	1485066691	Yuan	-132.3207, -34.0983	EI89
9	1485153091	Carlo	153.0054, 12.6429	129R
10	1487745091	Fatima	51.9435, 8.2735	49W4

The following code is to query a specific tuna that's been recorded:

```
// queryTuna - requires 1 argument
const request = {
chaincodeId:'tuna-app',
txId: tx_id,
fcn: 'queryTuna',
args: ['1']
};
return channel.queryByChaincode(request);
```

The code comes from `..src/queryTuna.js`.

Now, let's query for a specific tuna catch. This function takes one argument, as you can see in line 6 of the code. An example would be `['1']`. In this example, we are using the key to query for catches. You should see the following query response, detailing the attributes recorded for one particular catch:



Timestamp	Holder	Catch Location (Longitude, Latitude)	Vessel
1504054225	Miriam	67.0006, -70.5476	923F

The following code is to change the tuna holder:

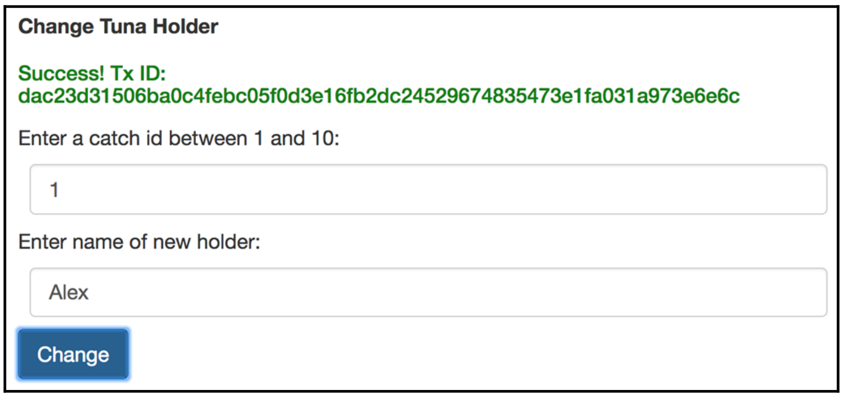
```
// changeTunaHolder - requires 2 argument
var request = {
chaincodeId:'tuna-app',
fcn: 'changeTunaHolder',
args: ['1', 'Alex'],
chainId: 'mychannel',
txId: tx_id
};
return channel.sendTransactionProposal(request);
```

The code comes from `..src/changeHolder.js`.

Now, let's change the name of the person in possession of a given tuna. This function takes two arguments, the key for the particular catch and the new holder, as we can see on line 5 in the preceding code, for example, `args: ['1', 'Alex']`. You may be able to see a similar success response in your Terminal window:

```
The transaction has been committed on peer localhost:7053 event
promise all complete and testing complete Successfully sent
transaction to the orderer. Successfully sent Proposal and
received ProposalResponse: Status - 200, message - "OK",
metadata - "", endorsement signature: 0D 9
```

This indicates that we have sent a proposal from our application through the SDK, and that the peer has been endorsed and committed and the ledger has been updated:



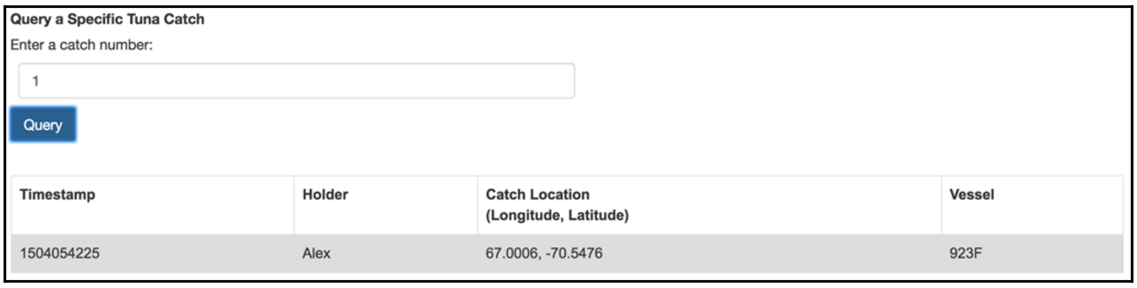
Change Tuna Holder

Success! Tx ID: dac23d31506ba0c4febc05f0d3e16fb2dc24529674835473e1fa031a973e6e6c

Enter a catch id between 1 and 10: 1

Enter name of new holder: Alex

You should see that the holder has indeed been changed by querying for key `1` again. Now, the holder attribute has been changed from `Miriam` to `Alex`:



Query a Specific Tuna Catch

Enter a catch number: 1

Timestamp	Holder	Catch Location (Longitude, Latitude)	Vessel
1504054225	Alex	67.0006, -70.5476	923F

- **Finishing up**: Remove all the Docker containers and images that we created in this tutorial with the following commands in the `tuna-app` folder:

```
$ docker rm -f
$(docker ps -aq)
$ docker rmi -f $(docker images -a -q)
```

Summary

We explored a new breed of blockchain Hyperledger project, built to focus on business challenges and overcome the distributed ledger technology. Hyperledger is the only group project led and hosted by the Linux Foundation that is on a continuous roadmap to revolutionize business with the distributed ledger-as-a-service model. This project helps the industry to avoid difficulties in deploying blockchains, just like WordPress solved the difficulty of turning up a website and Apache solved the problem of turning up a database.

In the next chapter, we will understand how blockchain technology can impact on existing and traditional security models, named **Confidentiality, Integrity, and Availability** (**CIA**) triad models.

Questions

With the tuna application, we have successfully understood the workings and testing of Hyperledger Fabric. However, to solve the cyber security challenges, it is important that we explore existing solutions and how they can be made much better with distributed ledger technology or generic blockchains, or maybe even with Hyperledger projects. Therefore, it is important to cover the following points:

1. Can Hyperledger Fabric be considered for a public blockchain?
2. Can Hyperledger be connected with a traditional database?