CASE STUDY

BLOCKCHAIN IN HYPERLEDGER: BETTER THAN ETL?

OVERVIEW

A technical walkthrough of a permissioned reference blockchain implemented with Hyperledger Fabric with focus on the potential value for enterprise-level organizations.

TECHNOLOGIES

Blockchain, Hyperledger Fabric, Go, Node.js, Go, Docker

KEYHOLE SOFTWARE

Introduction To Case Study

In general terms, a blockchain is an immutable transaction ledger in a distributed network of participating peers. Its data includes a string of transaction records secured with cryptography. Benefits of blockchain can include decentralization, immutability, provenance, and finality.

While Bitcoin and Ethereum cryptocurrencies brought blockchain to the forefront of technology headlines, the technology underneath has true potential value for the enterprise outside of the cryptocurrency space. The features provided by blockchain technology can lead business benefits like lower costs, higher efficiency, and lower risk. Seeing a technology actually applied reinforces understanding. It can also be a genesis for new ideas.

In this case study, we walk through a Hyperledger Fabric reference blockchain with a focus on showing the potential value for enterprise-level organizations.

We assume that you, as the reader, have a fundamental understanding of blockchain technology. If you do not, here is a link to our white paper on the topic: http://bit.ly/KHSblockchainWP.

The potential cost savings of blockchain is one of the benefits not really discussed as a whole. However, we at Keyhole believe it could be a significant feature benefit. The goal of this case study is to help reinforce this.

Introducing Hyperledger Fabric

Use cases in the enterprise are very different than in the cryptocurrency world. One glaring difference is the need for the enterprise to know the participants of the blockchain, particularly when financial or governmental regulations must be followed. This is a key difference: public versus permissioned networks. While many existing blockchain technologies have been adapted to fit enterprise use, there is a benefit to using a platform designed to be used by the enterprise.

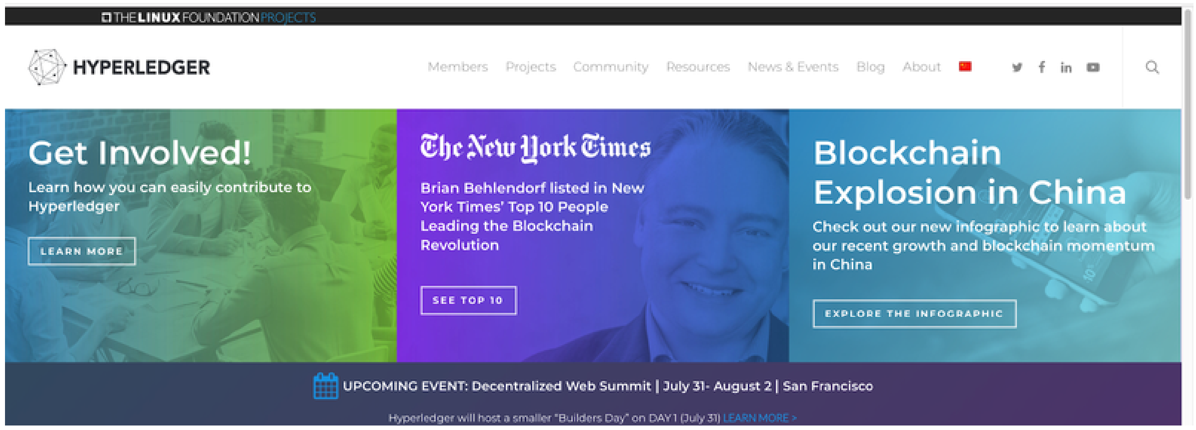
Hyperledger is an umbrella project of open source blockchains and related tools with a number of frameworks for distributed ledgers underneath. It is a modular, pluggable blockchain framework supported by the Linux Foundation.

Hyperledger Fabric is the Hyperledger project used for our blockchain implementation. Hyperledger Fabric provides the infrastructure to implement “permissioned” blockchain networks with custom consensus mechanisms. The framework is written in Google’s Go language and is highly configurable.

Hyperledger Fabric

Fabric supports the idea of “Smart Contracts” by allowing programming modules, written in Go, Java, or JavaScript and is referred to as “chaincode.” Chaincode modules are installed and executed within blockchain transaction invocations and is the mechanism for applying business rules and logic to the data that is stored in the blockchain.

Also in Hyperledger Fabric, multiple ledger “Channels” can be defined and managed, supporting a secure distributed environment for sharing data. For security, PKI Digital certificates and keys are required to access the network and query or create transactions.



Get Involved!

Learn how you can easily contribute to Hyperledger

The New York Times

Brian Behlendorf listed in New York Times' Top 10 People Leading the Blockchain Revolution

Blockchain Explosion in China

Check out our new infographic to learn about our recent growth and blockchain momentum in China

UPCOMING EVENT: Decentralized Web Summit | July 31- August 2 | San Francisco

Hyperledger will host a smaller "Builders Day" on DAY 1 (July 31) LEARN MORE >

The blockchain offers a way for groups to securely share common data transactions in a tamper-proof and secure manner. The true value is seen when used by groups like consortiums, associations, industries, and supply chains.

ETL Thought Experiment

Here’s a thought experiment to consider: think of the ETL (Extract Transform, Load) processes that your organization has in place. Now, specifically focus on the processes that work to receive data and process it into the line of business systems with the end goal of providing a way to publish that data for others to consume (i.e. not cubing data for reporting or analytic purposes).

Then consider the resources required to build and support those ETL processes: software packages, data stores, batch processing mechanisms, and don’t leave out the human resources required to code/perform the processing.

These activities are done differently by all parties and can result in the same data being copied repeatedly. Significant expense and energy are spent processing this data.

Applying Distributed Blockchain Technology with Hyperledger

With this ETL reference in mind, let's apply distributed blockchain technology.

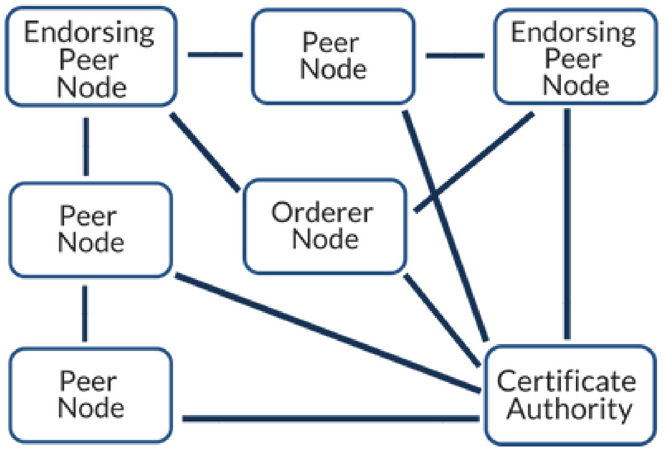
In a permissioned blockchain, a multitude of organizations and users participate in an authenticated manner. Hyperledger Fabric provides a MSP (Membership Service Provider) service to support this. Each organization that participates in the blockchain network is issued a digital certificate for access to the network.

Public/private keys are also generated for each organization and its users. These keys are used to digitally sign transactions, identify/authenticate themselves, and spin up a HyperLedger Peer on-domain address and port.

All user identities and addresses are known in a permissioned blockchain. This information is stored in the genesis block of the blockchain. Blockchains are appended for everything but this network meta-data stored in the genesis block.

When new nodes are added or removed, the Hyperledger network updates the genesis block located on all Peer Nodes. Peer Nodes can create Smart Contract transactions ("chaincode" in Hyperledger) that go through the consensus mechanism ("Orderer Nodes" in Hyperledger). If valid, all Peer Nodes are updated, confirmed, and a consensus is achieved.

Here is a diagram illustrating the participating nodes in a Hyperledger peer-to-peer network:



Each organization participating in the blockchain network will be a Peer or Endorsing Peer Node with the ability to create ledger transactions. However, Endorsing Peers have the authority to execute chaincode, which is a part of Hyperledger's consensus mechanism.

Compared to the ETL scenario from earlier, what we end up with is a network of authenticated peers adding and consuming the same immutable data source. There is no data duplication or need for heavy processes to get and add new transactions to the chain. This is a read and add only data source – no updates or deletes. This allows a strong level of confidence in the data and allows code that is written to be hardened around that. Instead of an extract, transform and load process – you simply pull the data you need from the stack and consume it downstream.

Consensus

Cryptocurrency: Proof of Work Consensus

Cryptocurrency frameworks tend to use a proof of work (PoW) type of consensus. In PoW, block miners compete for ownership of transactions in the network, forming a block with them, then sending them to all peers in the network for execution and validation of the blocks. The PoW-type consensus requires chaincode to be implemented in a deterministic language since the execution and results must always end up with the same result.

Hyperledger Consensus

Alternatively, the Hyperledger consensus mechanism is driven by the Orderer Service. The Orderer Service will gather transactions, then a subset of Endorsing Peers will execute the chaincode. If valid, they will order the transactions in a block and disseminate to the Peer Nodes, where the block will be validated and applied to the blockchain.

Hyperledger's approach allows a non-deterministic language like Java, Go, or JavaScript to implement chaincode. It also scales better than a PoW consensus mechanism. It's worth noting that Hyperledger's consensus algorithm is pluggable, so a custom consensus mechanism can be used.

Chaincode is programming logic that is installed and executed on a Channel (ledger). Chaincode procedures accept parameters specified by the Peer Node/User that request a transaction be executed. (Yes, a stored procedure is an appropriate analogy.)

When chaincode is invoked in a transaction, it will read and write data from the world state database. By default this is CouchDB. This is not the blockchain datastore, its purpose is to hold the latest data values for key IDs in the blockchain. All the blocks in the chain are stored on the file system in blockchain format. Transactions store a read/write set of values of the chaincode invocation.

This is a super simple and brief overview, but the idea is powerful. All entities participating in the network will be able to read and write data transactions in "near" real time. It's not real time due to the consensus mechanism taking time to bring the network into "consensus," but it can be fairly quick. It's certainly significantly quicker than running an ETL process at the end of the day or on some time interval.

Case Study

Blockchain in Hyperledger

Reference Blockchain Use Case Example

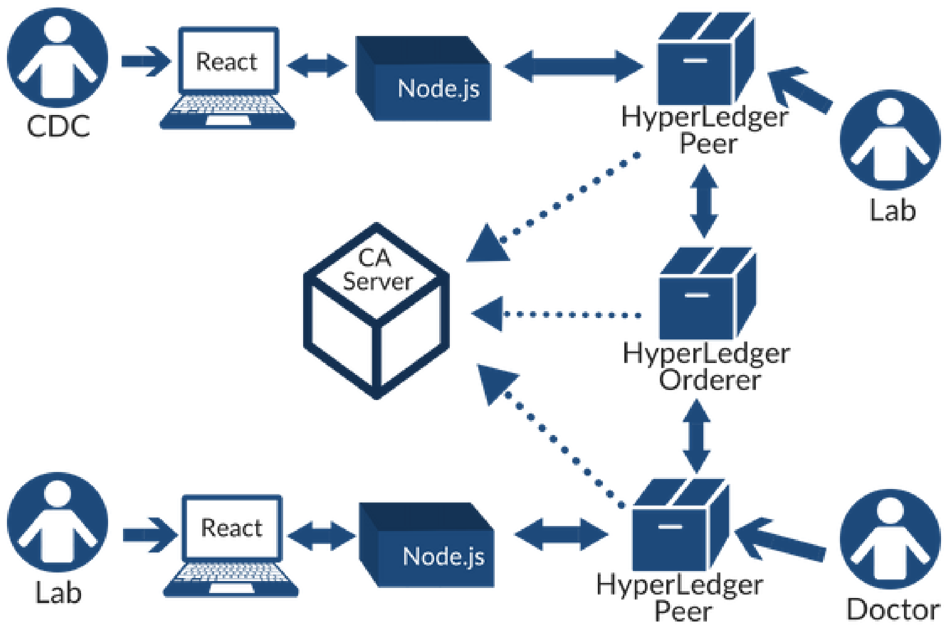
Management of the Flu

Potential blockchain use cases are far reaching across many industries and domains. We chose a case domain example that most people want positive management of, the flu.

Each year Influenza begins in the Southern Hemisphere and migrates around the world to the United States mutating along the way. The CDC makes educated guesses as to which strain will be the most virulent for which to create vaccines. When the flu season makes its way to the United States, the CDC and state-level agencies track its progress by medical practitioner reports and monitoring search engines for folks searching for the flu.

Imagine that a blockchain exists and the CDC, State Health and Human Services, medical practitioners, and testing labs are all participating as Peer Nodes in that blockchain.

Here is a diagram of the blockchain and an accessing React/Node client application:



This is the Hyperledger-based blockchain that we have implemented here at Keyhole Software.

Each participant starts up a Peer Node. Hyperledger provides a Docker-based container to start up a Peer. The Peer requires cryptographic keys, certificates, and for Hyperledger to provide a utility to generate these. A Certificate Authority is also required to authorize these certificates in the network, again this is provided by the framework.

Once started, a Channel can be created, which is a ledger. Then the Smart Contract ("chaincode" in Fabric speak) can be installed and instantiated on selected Peers that are designated as Endorsing Peers.

Tel. 877-521-7769
asktheteam@keyholesoftware.com
https://keyholesoftware.com

Chaincode (i.e. Smart Contracts)

Chaincode is the application logic of blockchain and is implemented in Go. It's a single module defined in a single file that is compiled into a binary. It is then installed on a Hyperledger Peer Node.

Go is a statically-typed, compiled language. Since Hyperledger Peers are implemented with Go, they have facilities to execute Go chaincode binaries. In addition, Hyperledger also supports JavaScript and Java-based chaincode modules which greatly increases the potential for developer and team adoption.

For our example, the Influenza chaincode function is defined in a Labs.go file. A Fabric stub interface provides an API to put and get data from the blockchain global data store. The Go data model structure stored in the blockchain is shown to the right.

```
// Define the LabResult.
//Structure tags are used by encoding/json library
type LabResult struct {
   Gender   string `json:"gender"`
   DOB      string `json:"dob"`
   City     string `json:"city"`
   State    string `json:"state"`
   TestType string `json:"testtype"`
   Result   string `json:result"`
   DateTime string `json:datetime"`
}
...
```

```
/*
*  Invoke Smartcontract with arguments
*/
func (s *SmartContract) Invoke(APIstub shim.ChaincodeStubInterface) sc.Response {

   // Retrieve the requested Smart Contract function and arguments
   function, args := APIstub.GetFunctionAndParameters()
   // Route to the appropriate handler function to interact with the ledger appropriately
   if function == "initLedger" {
      return s.initLedger(APIstub)
   } else if function == "queryAllLabs" {
      return s.queryAllEntries(APIstub)
   } else if function == "queryStateResults" {
      return s.queryStateResults(APIstub, args[0])
   } else if function == "createLab" {
      return s.createLab(APIstub, args)
   }

   fmt.Println("args ", args)

   return shim.Error("Invalid Smart Contract function name.")
}
```

Notice that Go has built-in JSON serialization capabilities.

The **invoke** function is the chaincode entry point when a transaction is invoked. To the left is the invoke method implementation.

Chaincode (i.e. Smart Contracts)

The **invoke** function will call to an operation based upon the function name specified. Here's the implementation of the **queryAllLabs** function.

```
func (s *SmartContract) queryAllEntries(APIstub shim.ChaincodeStubInterface) sc.Response {

  startKey := ""
  endKey := ""

  resultsIterator, err := APIstub.GetStateByRange(startKey, endKey)
  if err != nil {
    return shim.Error(err.Error())
  }
  defer resultsIterator.Close()

  // buffer is a JSON array containing QueryResults
  var buffer bytes.Buffer
  buffer.WriteString("[")

  bArrayMemberAlreadyWritten := false
  for resultsIterator.HasNext() {
    queryResponse, err := resultsIterator.Next()
    if err != nil {
      return shim.Error(err.Error())
    }
    // Add a comma before array members,
    //suppress it for the first array member
    if bArrayMemberAlreadyWritten == true {
      buffer.WriteString(",")
    }
    buffer.WriteString("{\"Key\":")
    buffer.WriteString("\"")
    buffer.WriteString(queryResponse.Key)
    buffer.WriteString("\"")

    buffer.WriteString(", \"Record\":")
    // Record is a JSON object, so we write as-is
    buffer.WriteString(string(queryResponse.Value))
    buffer.WriteString("}")
    bArrayMemberAlreadyWritten = true
  }
  buffer.WriteString("]")

  fmt.Printf("- queryAllLabs:\n%s\n", buffer.String())

  return shim.Success(buffer.Bytes())
}
```

This gives you a taste of the code we have in the labs.go implementation. If you're interested in diving deeper, here's a link to a full implementation on Github: http://bit.ly/2Kp8VGC

Installing & Executing

Using the CLI, chaincode can be installed on a Peer Node channel. This operation will compile and build the labs.go module and then install it on the Peer Node. Here is the CLI Peer command that performs this:

```
cli peer chaincode install -n lab -v ${starttime} -p github.com/lab
```

With the chaincode installed on the Peer, it can be instantiated on the blockchain's Peer channel with the following CLI command:

```
cli peer chaincode instantiate -o orderer.example.com:7050 -C
mychannel -n lab -v ${starttime} -c '{"Args":[""]}' -P "OR
('Org1MSP.member','Org2MSP.member')"
```

An orderer and endorsement policy options are specified. Mechanics of this will be discussed in our next blog. But, for this blog, the chaincode is ready to be invoked.

Installing & Executing

Lab results are appended to the blockchain by issuing the following command using the Hyperledger CLI command shown below.

```
cli peer chaincode invoke -o orderer.example.com:7050 -C mychannel -n lab -c '{"function":"createLab","Args":
["F","03/01/1995","Topeka","RI","Mouth Swab","100","2018:07:03:10:00" ]}'
```

The **createLab** smart contract function is executed with arguments specifying an Influenza lab result test. This invokes a transaction that the blockchain network will process and all nodes will receive the new block.

Influenza lab results can be queried by invoking a query Smart Contract operation. This is also invoked using the CLI with the following expression:

```
cli peer chaincode invoke -o orderer.example.com:7050 -C mychannel -n lab -c '{"function":"queryAllLabs","Args":[""]}'
```

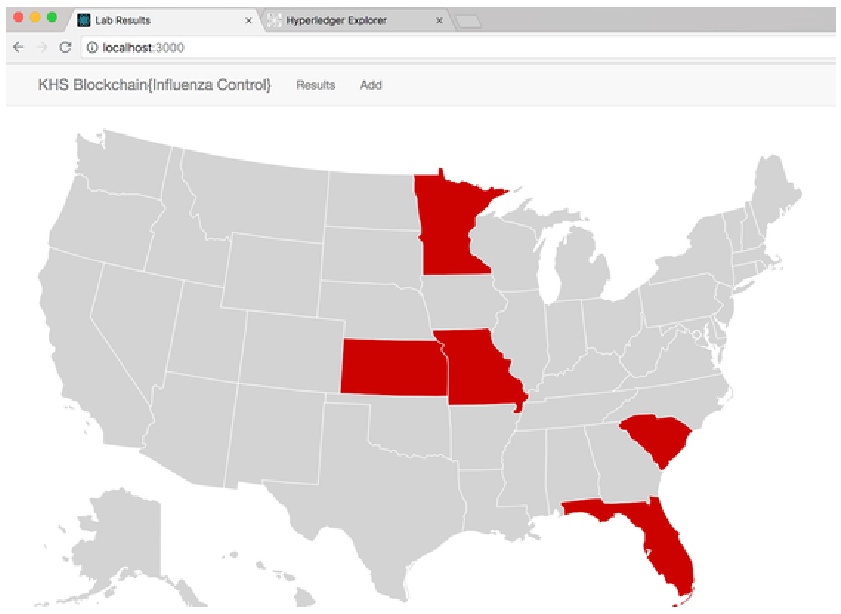
Notice the **queryAllLabs** function is invoked with no arguments specified. This will return all influenza results in the blockchain. Another query function can be invoked to return labs for a specific state by invoking the queryStateResults and specifying the state abbreviation as an argument. Complex and Compound query expressions can also be specified.

Integrating With a Web Application

With the Influenza blockchain being used by all parties, lab results will be distributed throughout the network applications. Blockchain users can interact and process this real-time data by creating client applications that access a Peer Node.

As an example of a client application, we use React for the user interface with Node.js as an API layer. Node.js uses the fabric-client.js module which allows access to a blockchain Peer.

A map of the U.S. displaying positive lab results can be rendered in a browser. This is depicted to the right.



Integrating With a Web Application

Hyperledger provides a Node.js module that allows a client to access and invoke chaincode on a Peer Node as long as it has a valid digital certificate. Here is a Node.js example that invokes the **queryAllLabs** Smart Contract to get results on behalf of the React.js user interface.

```
.......
 return;
}).then(() => {
  var transaction_id = client.newTransactionID();
  console.log("Assigning transaction_id: ", transaction_id._transaction_id);

  const request = {
      chaincodeId: 'lab',
      txId: transaction_id,
      fcn: 'queryAllLabs',
      args: ['']
  };
  return channel.queryByChaincode(request);
}).then((query_responses) => {
......
```

The expression to the left is defined in a Node.js server-side application that is called from an API call will return all test results as an array of JSON objects.

Lab tests can also be created and added to the blockchain using the fabric-client.js client. The following snippet is executed when the React.js UI calls a Node.js API that then executes this expression.

```
.......
   return;
}).then(() => {
  console.log("Make query");
  tx_id = client.newTransactionID();
  console.log("Assigning transaction_id: ", tx_id._transaction_id);

  // queryCar - requires 1 argument, ex: args: ['CAR4'],
  // queryAllCars - requires no arguments , ex: args: [''],
  console.log("labs" + JSON.stringify(lab));
  const request = {
      targets: targets,
      chaincodeId: 'lab',
      txId: tx_id,
      fcn: 'createLab',
      args: [ lab.gender, lab.dob, lab.city, lab.state, lab.testType, lab.result, lab.dateTime]

  };
  return channel.sendTransactionProposal(request);
....
```

The fabric-client.js node module provides an API to perform all the functions of the CLI (i.e. create a channel, install, and instantiate chaincode).

SUMMARY

BLOCKCHAIN IN HYPERLEDGER

By presenting an actual blockchain implementation of an easy-to-understand domain, we hope that you see the potential distributed information sharing capabilities of this technology.

It is important to note that blockchains are not going to replace traditional data stores. Even though Peer Nodes are backed by a data store, the data originating in a blockchain needs to be global data type values that have meaning across a domain or group.

Data such as customer IDs and overhead costs only have meaning within an organization. Transactional data that is shared among a group, consumption, or population of interested users is a candidate for living in a blockchain ledger.

If you want to implement your own proof-of-concept, or take a deeper dive into implementing a blockchain with Hyperledger, contact us at Keyhole Software. We have education and mentoring services that can assist your team in deploying blockchain technology.

KEYHOLE SOFTWARE

877-521-7769
https://keyholesoftware.com
asktheteam@keyholesoftware.com

Kansas City | St. Louis | Lincoln | Omaha | USA