EAS-IPM

Design Document – P2P Process Network

VERSION 1.0

	Prepared By / Last Updated By	Reviewed By	Approved By
Name	Nisha M. R	Anaswar G	Vinod Kumar Pai
Role	Developer	Sr. Developer	Associate Director
Signature			
Date	18-10-2019	18-10-2019	

Table of Contents

Introduction

1.1 Purpose

The purpose of this document is to provide an explanation of the Procure to Pay Network

1.2 Background

Procure-to-Pay(PTP) is the multi-step process blockchain solution that connects a client with one or more service/product providers. It also allows for the identification and authentication of stake holders, service provisioning, budgeting, invoicing, and payment settlement. The network tracks the flow of purchase order created by manufacturer till the consumption is done and the payment is received by supplier of the purchase order.

1.3 Requirements and Dependencies

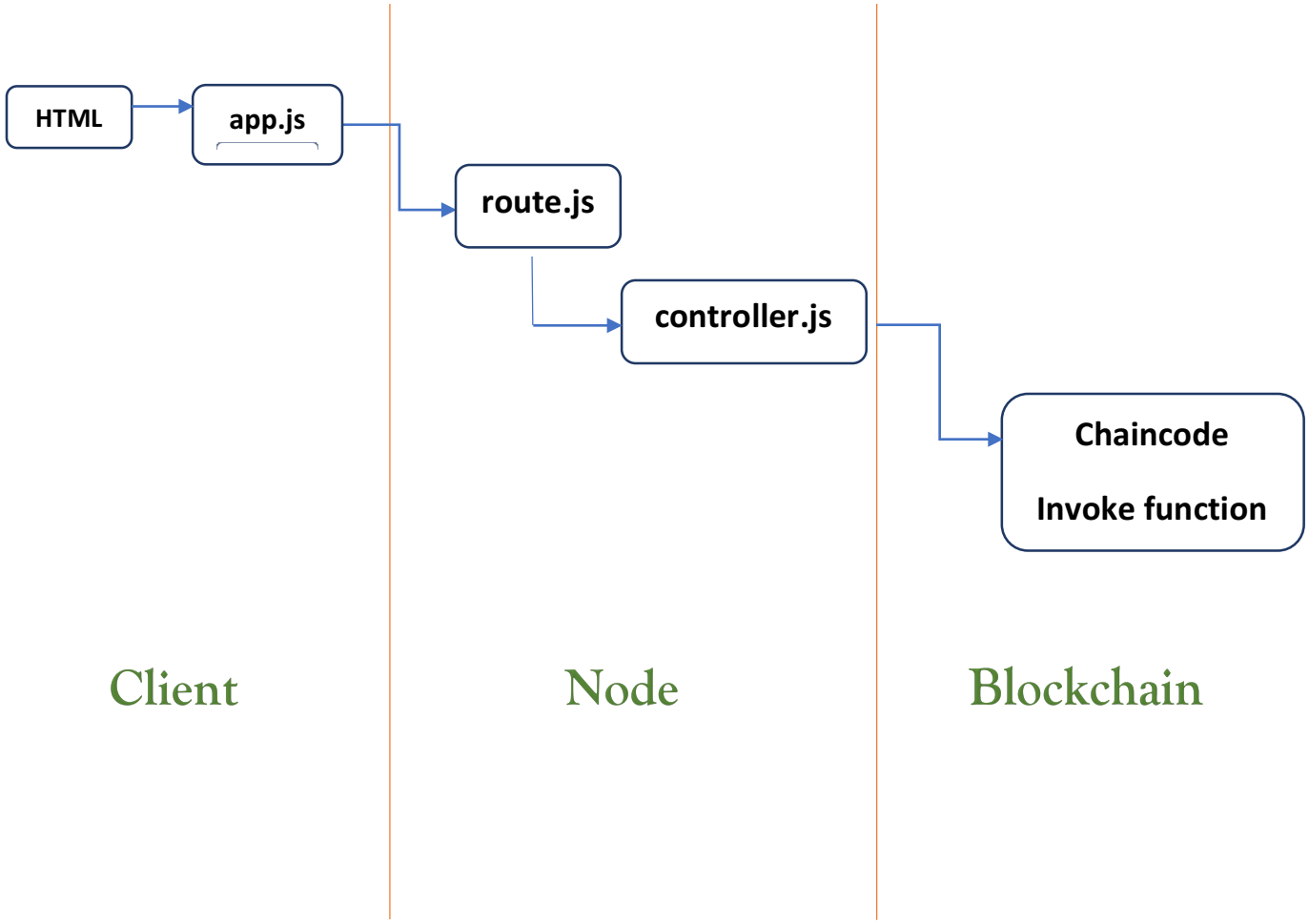
The following were used to build the application.

1) Hyperledger Fabric – 1.4
2) Node JS - v8.11.3
3) GoLang - go1.12
4) Angular JS
5) IDE – Visual Studio Code

1.4 Network Architecture

The application is divided into 3 layers.

1) User Interaction Layer
2) SDK Layer
3) Blockchain Layer



1.4.1 User Interaction layer

The user interaction layer includes a presentation layer. This layer is built using HTML/ Angular JS and CSS.

1.4.2 SDK Layer

This is the middle/interface layer which is the gateway to the blockchain. The middle layer provides the communication between user interaction layer and the blockchain layer. This layer is built using Node js. Java SDK is also available. The transaction submitted by the user is routed to controller file which is the node layer via a routes.js file. The controller file will have the orderer address and peer address which are participating in the network. The named of installed chaincode, the transaction and the arguments of the transaction is send to the chaincode in the form of request. The transaction is sent using sendTransactionProposal The

proposal response will be sent back to sdk by the chaincode and this response can be sent back to UI. For query transaction, the request is sent to chaincode using querybyChaincode.

Controller.js syntax

```
      module.exports = (function() {
      return{
    PlaceOrder: function(req, res){
        var channel = fabric_client.newChannel('mychannel');
        var peer1 = fabric_client.newPeer('grpc://localhost:7051');
        var peer2 = fabric_client.newPeer('grpc://localhost:8051');
        var peer3 = fabric_client.newPeer('grpc://localhost:9051');
        var peer4 = fabric_client.newPeer('grpc://localhost:10051');
        channel.addPeer(peer1);
        channel.addPeer(peer2);
        channel.addPeer(peer3);
        channel.addPeer(peer4);
        var order = fabric_client.newOrderer('grpc://localhost:7050')
        channel.addOrderer(order);

        const request = {
            //targets : --- letting this default to the peers assigned to the channel
            chaincodeId: 'p2pchaincode',
            fcn: 'placeOrder',
            args: [poNumber, materialCode, lineNumber, orderQuantity, uop, deliveryDat
      e, creationDate, price, currency, supplier],
            txId: tx_id
        };
        // send the transaction proposal to the peers
        return channel.sendTransactionProposal(request);

    }
}
});
```

Here peer1 and peer 2 belongs to org1 (Manufacturer) and peer3 and peer4 belongs to org2 (supplier). We have to specify the peer to which is acting as committing peer.

```
    //let event_hub = fabric_client.newEventHub();
    // event_hub.setPeerAddr('grpc://localhost:7051');
    let event_hub = channel.newChannelEventHub('localhost:7051');
```

For **search** transaction, target peer has to specified along with the request.

```
const request = {
    targets : [peer1],
    chaincodeId: 'p2pchaincode',
    txId: tx_id,
    fcn: 'search',
    args: [ID]
};
console.log(request)
// send the query proposal to the peer
return channel.queryByChaincode(request);
```

1.4.3 Blockchain Layer

In the blockchain layer we have the chaincode or smart contract. The chaincode is written in Golang. The call to chaincode is done through SDK. The explanation of chaincode and the functions implemented are explained in later section.

1.5 Application Participants

1) Manufacturer

2) Supplier

1.6 Application Description

There are two organizations in the network – Manufacturer & Supplier. In the network implementation, these two are defined as org1 and org2. The two organizations have two peer each of which one is the endorsement peer. The network is using orderer type of solo. A channel named “mychannel” is created and all the four peers are joined to the channel. There is a script file to start the hyperledger fabric network and to install the chaincode. While running the script file the peer, chaincode, orderer and CA containers are created.

For running our chaincode on the fabric network there are a series of steps to be followed.

In the utils.sh the commands to install, instantiate and invoke the chaincode needs to be specified for before running the network.

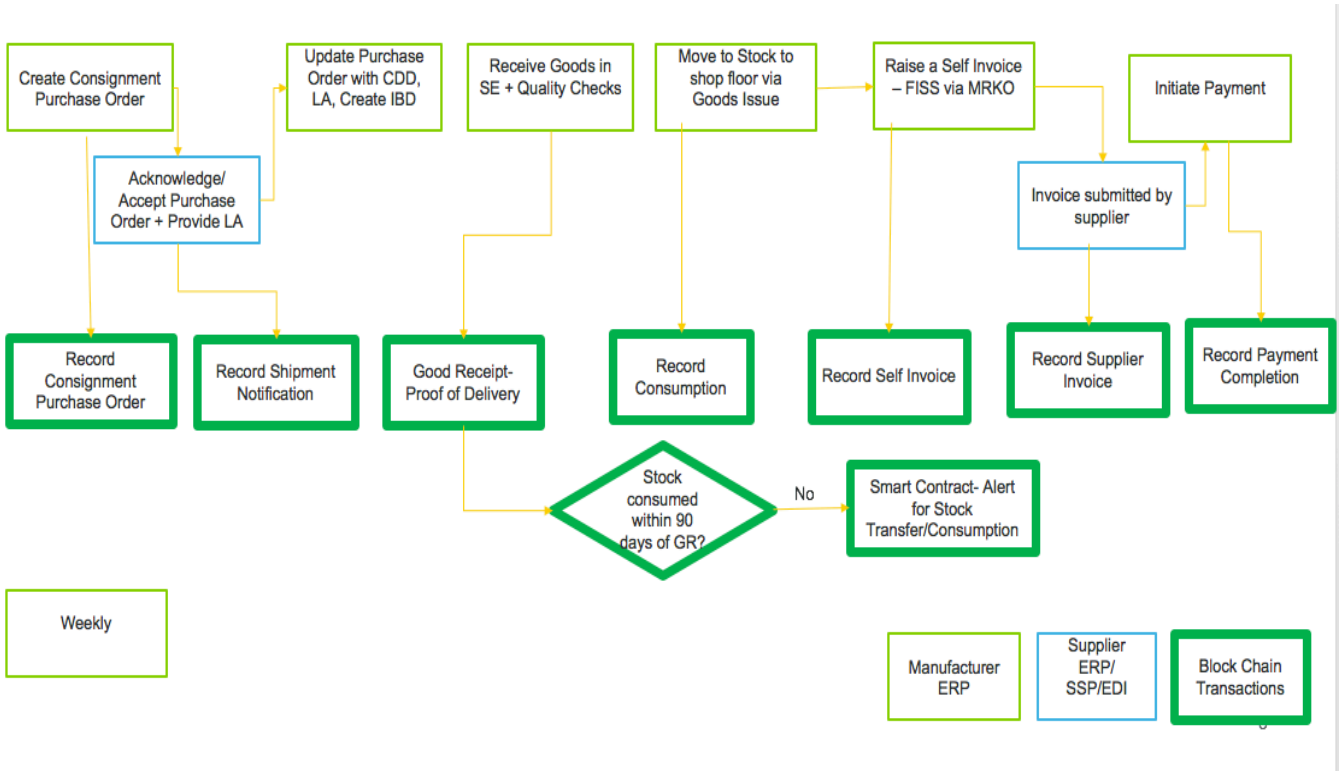
Syntax:

```
peer chaincode install -n p2pchaincode -v ${VERSION} -l ${LANGUAGE} -p ${CC_SRC_PATH} >&log.txt
```

```
peer chaincode instantiate -o orderer.example.com:7050 -C $CHANNEL_NAME -n p2pchaincode -l ${LANGUAGE} -v ${VERSION} -c '{"Args":["init"]}' -P "OR ('Org1MSP.member','Org2MSP.member')" >&log.txt
```

```
peer chaincode invoke -o orderer.example.com:7050 -C mychannel -n p2pchaincode $PEER_CONN_PARMS -c '{"Args":["placeOrder","M1","PO1","L1","100","DOZEN","10-10-2-18","10-09-2018","1000","USD","FLEXTRONICS"]}' >&log.txt
```

1.7 Flow Diagram



1.8 Chaincode

The chaincode is written in Golang. The packages needed for implementation are imported from "Shim". The important functions defined for the chaincode are

a) Init

Init function is called during initialization or updte done on the chaincode. Init function takes ChaincodeStubInterface as parameter. When we call this function from CLI, it responds back with the peer response.

```
func (t *ProcureToPayChaincode) Init(stub shim.ChaincodeStubInterface) pb.Response {}
```

b) Invoke

All the logic implementation functions are written here. These implementation functions are called transactions. There are insert, update and query transactions. The transaction list is explained later.

Syntax:

```
func (t *ProcureToPayChaincode) Invoke stub shim.ChaincodeStubInterface) pb.Response {

function, args := stub.GetFunctionAndParameters()

if function == “placeOrder”{

return t.placeOrder(stub, args)

}

return shim.Error(“Error invoking function”)

}
```

1.8.1 Assets

The following assets are defined in the chaincode.

1) Material

ID-material code

The material for which the purchase order is created. Identified by material code. Purchase order details will be stored in material in the form of array

2) Purchase Order

ID-PO Number

Purchase order contains the details of the specific order for a particular material. Identified by PO number. Batch details will be stored in this asset in form of array.

3) Batch

ID-Batch Code

The details of the batch which is shipped, received or invoice generated is stored in this asset. Identified by batch code.

4) Invoice

ID – Invoice Doc Num

The details of invoice generated by the manufacturer when a consumption is done or the invoice generated due to overdue (90 days in stock) is stored in invoice. Identified by invDocNum.

1.8.2 Transactions

1) PlaceOrder

   The manufacturer places an order to supplier or they can submit bulk orders. Purchase Order asset is created and also the purchase order details is stored in material asset.

2) shipmentNotification

   Supplier supplies the order in batches to manufacturer.

3) goodsReceipt

   Manufacturer updates the material, batch and purchase order assets when the batch is received

4) recordConsumption

   Manufacturer consumes the product and the invoice is generated

5) generateInvoice

   If the difference in number of dates between current date and date of receiving the batch is greater than 90, the notification will be sent to manufacturer and s supplier. Manufacturer then generates the invoice for that batch. This functionality is implemented in this transaction.

6) invoiceStatus

   This transaction is for changing the status of invoice when invoice is approved by supplier and payment is completed by manufacturer. The invoice generated by manufacturer has to be approved by supplier before sending the invoice for payment. After approving the invoice, it is send for payment.

7) Search

   This transaction is to search the assets using their identifier.

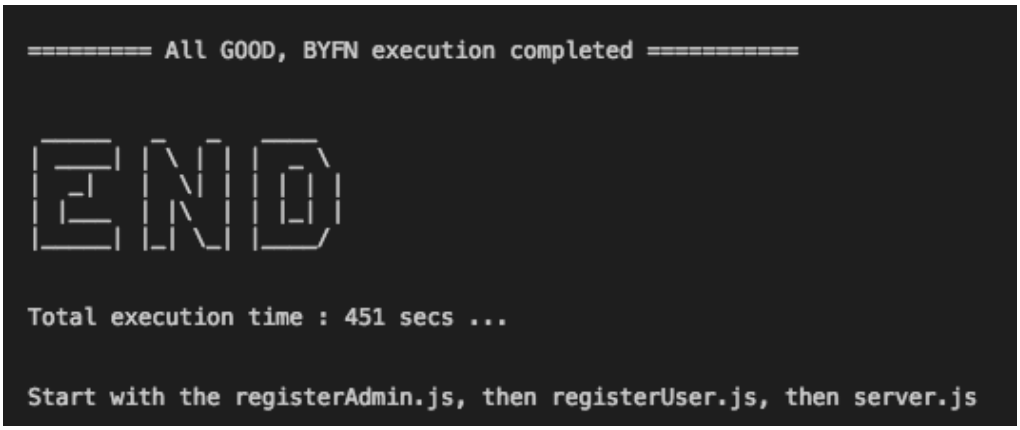
1.9 Running Network

For running the application, we have to start the fabric network, install the chaincode and then the node SDK is started. The following steps have to be followed.

1) Navigate to the folder which contain the shell script for starting the fabric. Here there is a startFabric.sh script. When we run this script, it will call the byfn.sh script in the network folder.

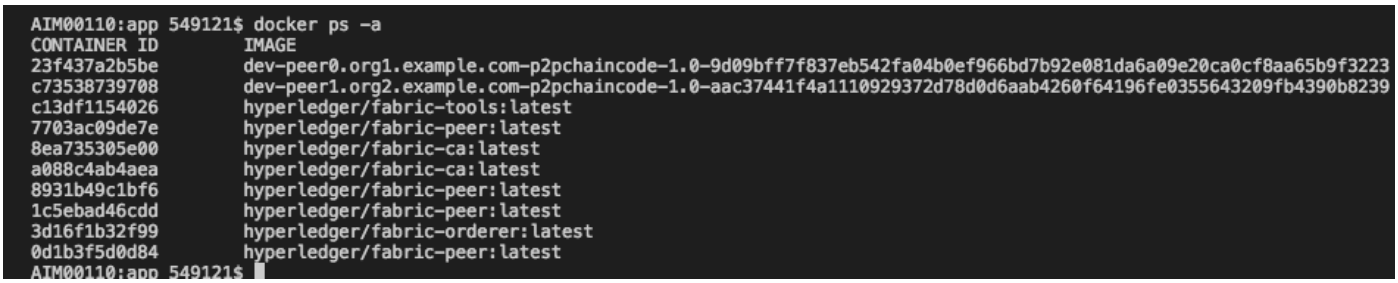
./startFabric.sh

If this script is run successfully, the following result will be displayed.



2) We can see the list of containers that is up and running.

docker ps -a



Now the chaincode is installed and instantiated in all the peers of the channel.

3) Node modules

The required node modules need to be installed. This is done using command

npm install

This will install the packages that are predefined in package.json. If any error specifying package is not available, we can install that using command npm install.

4) **registerAdmin.js** and **registerUser.js**

The admin and users have to be enrolled for each ca. In the network since this is multi org, there are two ca containers and using the certificate file of ca we have to enroll admin and user.

For enrolling admin

node registerAdmin.js

For enrolling user

node registerUser.js

This will create a **hfc-key-store** in the SDK folder and create the user and admin. This is later used by controller when committing a transaction.

5) Now we have to start the node app

Go to the root folder and hit the following command

npm start OR **node server.js**

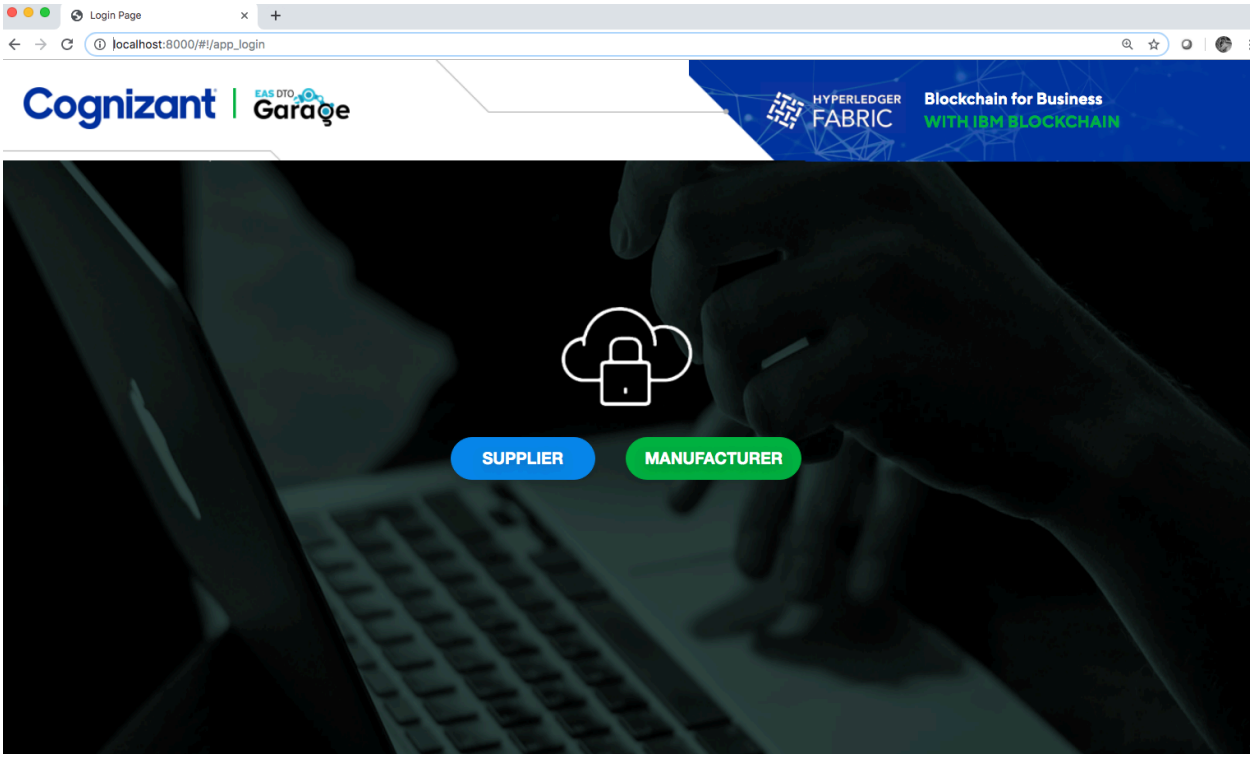
The following will be displayed if the app is started successfully

```
AIM00110:ProcureToPay 549121$ npm start

> Fabric-ElectriCharge-app@1.0.1 start /Users/549121/Desktop/PROCURETOPAY-FABRIC/ProcureToPay
> node server.js

/Users/549121/Desktop/PROCURETOPAY-FABRIC/ProcureToPay
Live on port: 8000
```

We can hit the application at http://localhost:8000/


Login Page
localhost:8000/#!/app_login
Cognizant
EAS DTO Garage
HYPERLEDGER FABRIC
Blockchain for Business
WITH IBM BLOCKCHAIN
SUPPLIER
MANUFACTURER