

**EAS-IPM**

**Design Document – Track & Trace Network**

**VERSION 1.0**

****

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Prepared By / Last Updated By** | **Reviewed By** | **Approved By** |
| **Name** | Nisha M. R |  | Vinod Kumar Pai |
| **Role** | Sr. Developer |  | Associate Director |
| **Signature** |  |  |  |
| **Date** | 25-10-2019 |  |  |

# Introduction

* 1. **Purpose**

The purpose of this document is to provide an explanation of the track and trace Network

* 1. **Background**

Track and trace network tracks the movement of a product from manufacturer shipped through distributor and wholesaler till it reaches hospital. Network tracks the product status and location.

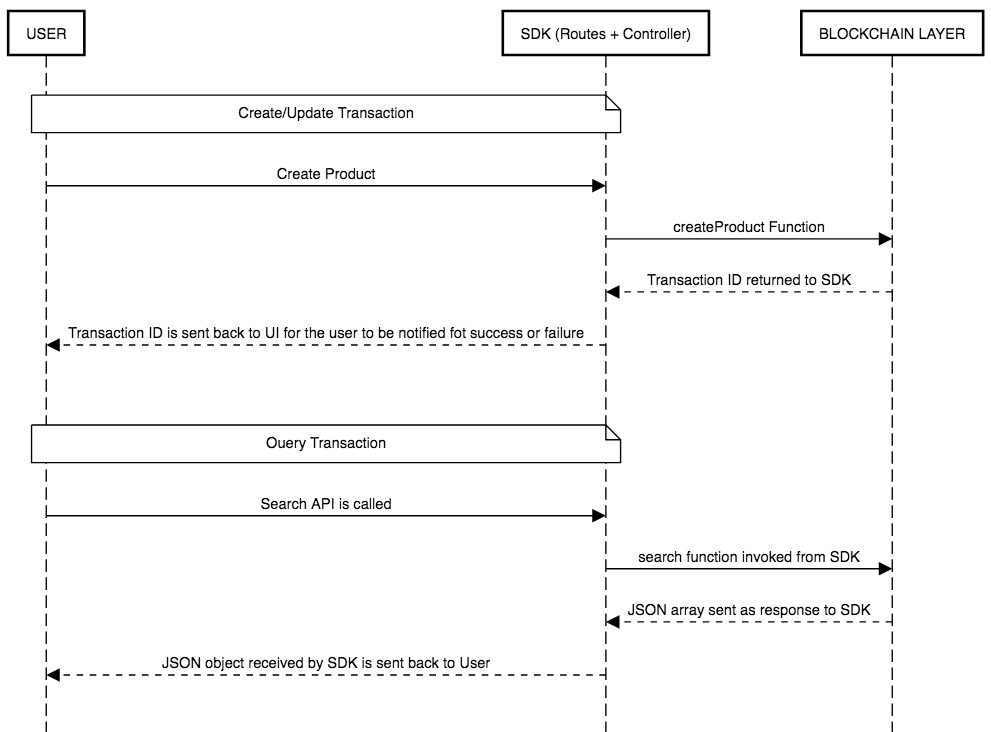
* 1. **Requirements and Dependencies**

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. **Network Architecture**

The application in divided into 3 layers.

1. User Interaction layer
2. SDK Layer
3. Blockchain Layer



* + 1. **User Interaction Layer**

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

* + 1. **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{

    createProduct: 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);

var input = req.body

        var drugID = input.drugid;

        var drugName = input.drugname;

        var drugStatus = input.drugstatus;

        var manufacturer = input.manufacturer;

        var components = input.components;

        var argument = [drugID, drugName, manufacturer, drugStatus, components];

         const request = {

                //targets : --- letting this default to the peers assigned to the channel

                chaincodeId: tracktracechaincode,

                fcn: createProduct,

                args: args: [uuid, material, make, rawMaterialLocation, productStatus, shipmentStatus, batchCode],

                //chainId: 'mychannel',

                txId: tx\_id

            };

        // send the transaction proposal to the peers

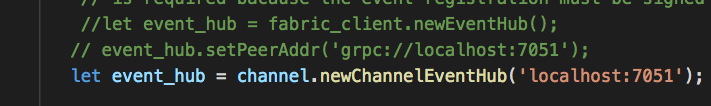
        return channel.sendTransactionProposal(request);

    }

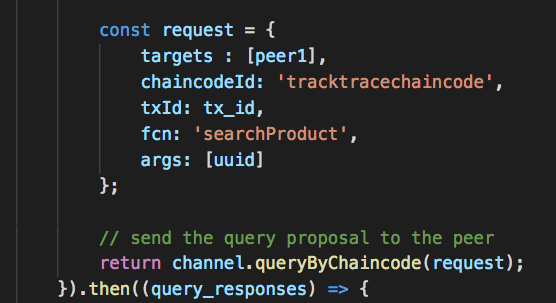
}

});

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



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



* + 1. **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. **Application Participants**

1. Manufacturer
2. Distributor
3. Dealer
4. Hospital
   1. **Application Description**

There are four participants in the network as mentioned in Application participants section.

The manufacturer registers the product. The created product will be available in the ledger.

The product traverses through the distributor and delaer and finally reaches the hospital.

IoT device is used to record the temperature. The temperature can be recorded at any point of transport. If the temperature is greater than **25(threshold limit),** the product will be tampered and it will not be shipped further.

In each of the organizations, one pair is defined as anchor peer. A channel named “mychannel” is created and the all the peers are added 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.

The commands to install, instantiate and invoke chaincode needs to be specified in the shell script file in the first network of fabric-samples (utils.sh).

Syntax:

peer chaincode install -n tracktracechaincode -v ${VERSION} -l ${LANGUAGE} -p ${CC\_SRC\_PATH} >&log.txt

 peer chaincode instantiate -o orderer.example.com:7050 -C $CHANNEL\_NAME -n tracktracechaincode -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 tracktracechaincode $PEER\_CONN\_PARMS -c '{"Args":["createProduct","1","Scanner","Siemens","Pune","IN\_GOOD\_CONDITION","IN\_PRODUCTION","1\_Batch"]}' >&log.txt

* 1. **Flow Diagram**

MANUFACTURER

Create Product

If IoT temp > 25

IOt updates temperature

DISTRIBUTOR

PRODUCT TAMPERED

No Shipment status Update

IOt updates temperature

DEALER

IOt updates temperature

HOSPITAL

* 1. **Chaincode**

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

1. Init

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

//Init - initialisation of chaincode

func (t \*TrackTraceChaincode) Init(stub shim.ChaincodeStubInterface) pb.Response {

    // nothing to do

    return shim.Success(nil)

}

1. 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:

//Invoke - Invoke functions of chaincode

func (t \*TrackTraceChaincode) Invoke(stub shim.ChaincodeStubInterface) pb.Response {

    function, args := stub.GetFunctionAndParameters()

    if function == "createProduct" {

        return t.createProduct(stub, args)

    }

    return shim.Error("Error invoking function")

}

* + 1. **Assets**

The following assets are defined in the chaincode.

1. Product

**ID – UUID**

The product details are stored in this asset. The asset is updated when any updating is done on the asset.

* + 1. **Transactions**

1. **createProduct**

New product is created for unique UUID. The product asset is created in this transaction

1. **updateShipmentStatus**

The product that is created is transported by distributor and dealer and it reaches the final state. This transaction is done invoked to update the ledger

1. **updateProductStatus**

When the IoT device updates the temperature, this transaction is invoked.

1. **searchProduct**

This transaction is used to query the assets using the product UUID. Target peer will be specified at the time of querying.

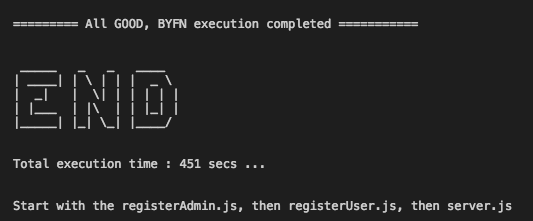
* 1. **Running Network**

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.

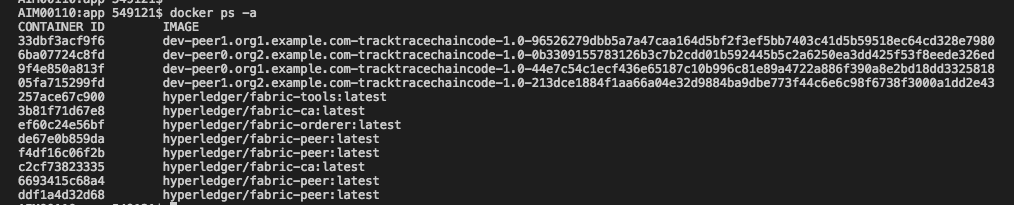
**./startFabric.sh**

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



1. We can see the list if containers that are up and running using the following command.

**docker ps –a**



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

1. 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.

1. **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**

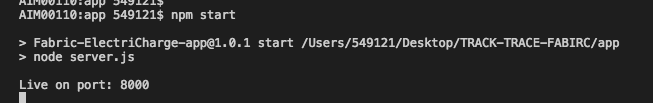
This will create a **hfc-key-store** in the SDK folder and create the user and admin. This is later used by controller when committing a transaction.

1. 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.



We can hit the application using url: <http://localhost:8000/>

