Distributing Trust & Blockchains

Assignment 3

Team 13 (Tripple):

Tathagato Roy (2019111020)
Snehal Kumar (2019101003)
Rutvij Menavlikar (2019111032)

Hyperledger Fabric

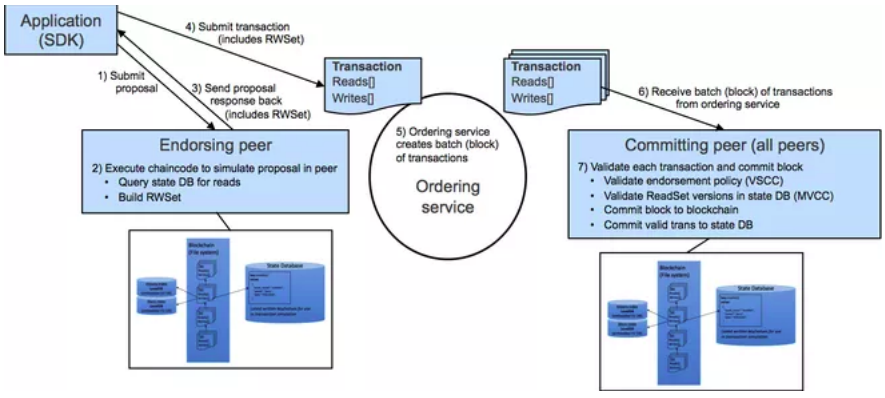
Hyperledger Fabric is an permissioned distributed ledger framework used for developing solutions and applications launched by IBM. It is primarily intended for enterprises and private permissioned use and for that purpose it has been designed in a modular and versatile fashion in order to satisfy different use cases in the industry. Fabric can be divided into three main components:

- Smart contracts called chaincode
- Transaction Order
- Transaction Commitment

The division in this manner and the flow of network gives it high performance with better scalability than other previous platforms. The network works following the process:

- A transaction is submitted to the endorser peer.
- The proposal is given to the committers after the endorser peers have given sufficient endorsements according to the endorsement policy.
- The committers validate the each transaction and verify that the peers have followed the endorsement policy.
- The transaction is committed to the ledger after the committers have successfully validated the transaction

The detailed view can be seen in the figure:



Hyperledger allows the creation of channels to maintain privacy within the network and only the nodes in the channel can access the information in that channel. For chaincode, Fabic supports Go, Javascript and Java languages.

Implementation

To build the required network and simulate the transactions, we provide a `run.sh` file in the `test-network` folder which contains all the commands to setup and simulate the network and an auction. To run the script go to the `test-network` folder and use the command:

```
./run.sh
```

There are three chaincodes given in the `assignment3` folder which contains the implementation of each team member.

To build a basic fabric network, we use the `test-network` directory in the fabric-samples. To start the network and add a channel we use the command:

```
./network.sh up createChannel -c assgn3
```

We then add a third org using the script in `addOrg3` folder.

```
./addOrg3.sh up -c assgn3
```

Then we need to install the chaincode on all the peers for endorsement for which we first need the peer CLI which is done by adding its binaries to the path and installing the package by:

```
peer lifecycle chaincode package <package-name> --path <path-to-chaincode> --lang <language> --label <label>
```

Once the packages have been installed on the peers, we approve them by using the orderer and the package ID.

```
peer lifecycle chaincode approveformyorg -o <orderer-address> --ordererTLSHostnameOverride orderer.example.com --channelID assgn3 --name <p
```

Then we have to commit the chaincode:

```
# Commit the chaincode
peer lifecycle chaincode commit -o <orderer-address> --ordererTLSHostnameOverride orderer.example.com \
  --channelID assgn3 --name <package> --version 1.0 --sequence 1 --tls --cafile <orderer-certificate> \
  --peerAddresses <first-peer-address> --tlsRootCertFiles <tls-certificate> --peerAddresses <second-peer-address> \
  --tlsRootCertFiles <tls-certificate> --peerAddresses <third-peer-address> --tlsRootCertFiles <tls-certificate>

# Check if query has been committed
peer lifecycle chaincode querycommitted --channelID assgn3 --name <package> \
    --cafile <orderer-certificate>
```

Now we have done the setup required and have to simulate the auction which is done by using the invoke command on the peer cli. First we create an auction, then using each peer, make a bid for the listing and finally declare the winner of the auction.

```
peer chaincode invoke -o <orderer-address>> --ordererTLSHostnameOverride orderer.example.com --tls --cafile \
  <orderer-certificate> -C $CHANNEL -n <package> \
  --peerAddresses <first-peer-address> \
  --tlsRootCertFiles <first-peer-cer> \
  --peerAddresses <second-peer-address> \
  --tlsRootCertFiles <second-peer-certificate> \
  --peerAddresses <third-peer-address> \
  --tlsRootCertFiles <third-peer-certificate> \
  -c '{"function":"createAuction","Args":["Asset1"]}'

peer chaincode invoke -o <orderer-address>> --ordererTLSHostnameOverride orderer.example.com --tls --cafile <orderer-certificate> \
  -C $CHANNEL -n <package> \
  --peerAddresses <first-peer-address> \
  --tlsRootCertFiles <first-peer-cer> \
  --peerAddresses <second-peer-address> \
  --tlsRootCertFiles <second-peer-certificate> \
  --peerAddresses <third-peer-address> \
  --tlsRootCertFiles <third-peer-certificate> \
  -c '{"function":"submitBid","Args":["Asset1", "300"]}'

peer chaincode invoke -o <orderer-address>> --ordererTLSHostnameOverride orderer.example.com --tls --cafile <orderer-certificate> \
  -C $CHANNEL -n <package> \
  --peerAddresses <first-peer-address> \
  --tlsRootCertFiles <first-peer-cer> \
  --peerAddresses <second-peer-address> \
```

```
  --tlsRootCertFiles <second-peer-certificate> \
  --peerAddresses <third-peer-address> \
  --tlsRootCertFiles <third-peer-certificate> \
  -c '{"function":"declareWinner","Args":["Asset1"]}'
```

The smart contract implementation consists of a simple auction class which contains functions for creating auction, submitting bid, declaring the winner. To communicate and utilize the client's certificate for decisions we use `ctx.GetStub().GetCreator()` where each function has an argument "ctx" (transaction context) which is used to access the stub.

The auction is initialised with no bids, winner and a status `ACTIVE`. Then using the script, we call the `submitBid` function to make a bid for each peer. Here we add the bid to the auction if it is a valid bid and not been made previously.
`ctx.stub.putState(assetID,Buffer.from(JSON.stringify(auction)))` is used to update the value. After placing all the bids, `declareWinner` calculates the highest bid made and assigns the asset to the corresponding bidder and change the state of auction to `ENDED` .

```
/// Calculate the highest bidder
    Object.keys(auction.bids).forEach((key) => {
      if (auction.bids[key] > winningPrice) {
        winningPrice = auction.bids[key];
        winner = key;
      }
    });
```

This logic is implemented in different ways in each of the implementations.