Bharat\_ecom1

First Network using Command Line

The steps we are going through are,

Generation of crypto/certificate using *cryptogen*

Generation of Configuration Transaction using *configtxgen*

Bring up the nodes based on what is defined in docker-compose file

Use CLI to setup the First Network

Use CLI to install and instantiate the chaincode

Use CLI to invoke the chaincode

Tear down all the setup

Preparation

As we may have something left after our previous setting, check the following.

no *crypto-config* directory in first-network

empty in *channel-artifacts* directory (but keep this directory)

no docker containers are running

// remove crypto-config directory

$ cd Ecommerce

$ rm -rf crypto-config/

// empty channel-artifacts directory

$ cd Ecommerce/channel-artifacts

$ rm \*

// kill and remove docker containers

$ docker kill $(docker ps -aq) // return error if no running containers

docker kill $(docker ps -aq)

docker stop $(docker ps -aq)

docker rm $(docker ps -aq)

docker system prune -a

docker volume prune

docker volume prune -f

docker network prune

$ docker rm $(docker ps -aq) // return error if no containers

$ docker ps // see no running containers

Step 1: Generation of crypto / certificate

The tool for crypto generation is called *cryptogen*, which comes as binary tool when first installing the *fabric-samples*.

We use the *crypto-config.yaml* to define the required setup.

$ cd Ecommerce

$ ../bin/cryptogen generate --config=./crypto-config.yaml

We see a new directory *crypto-config* is created, and inside there are directories corresponding to the table shown above. You can browse the sub-directories in each directory and see the directory hierarchy.

Step 2: Generation of Configuration Transaction

The tool for configuration transaction generation is called *configtxgen*, which also comes as binary tool when first installing the *fabric-samples*.

The artifacts to be generated in this step are,

orderer genesis block

channel configuration transaction

one anchor peer transaction for each peer organizations (total 2 in our example)

When we use command *configtxgen*, the *configtx.yaml* file is referred.

The two profiles defined here are,

TwoOrgsOrdererGenesis

TwoOrgsChannel

Four files are to be generated and stored in channel-artifacts directory. They are,

genesis.block

channel.tx

Org1MSPanchors.tx

Org2MSPanchors.tx

Command to generate the genesis block.

$ export FABRIC\_CFG\_PATH=$PWD

$ ../bin/configtxgen -profile TwoOrgsOrdererGenesis -outputBlock ./channel-artifacts/genesis.block

Command to generate the channel transaction (we need to specify the channel name *mychannel* here).

$ export CHANNEL\_NAME=mychannel

$ ../bin/configtxgen -profile TwoOrgsChannel -outputCreateChannelTx ./channel-artifacts/channel.tx -channelID $CHANNEL\_NAME

Command to generate the transaction for the Anchor Peer in each Peer Organizations

$ ../bin/configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate ./channel-artifacts/buyerMSPanchors.tx -channelID $CHANNEL\_NAME -asOrg BuyerMSP

$ ../bin/configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate ./channel-artifacts/sellerMSPanchors.tx -channelID $CHANNEL\_NAME -asOrg SellerMSP

After all commands are issued, there are four files created inside *channel-artifacts* directory.

Step 3: Bring up the nodes according to docker-compose file

With the crypto / certificates and the configuration transaction generated, we are now ready to bring up the First Network.

Command to bring up the containers

$ docker-compose -f docker-compose-cli.yaml up -d

If we check the running containers before and after the docker-compose commands (docker ps), we will see now six containers are up and running.

Step 4: Use CLI to setup the First Network

From now on, we are going to build and interact with the First Network. All commands will be issued inside CLI when configuring all components for First Network.

Access shell of CLI container from localhost is,

$ docker exec -it cli bash

Important Note: the CLI tool has default setting on Peer0 of Org1 node. Which means, without specifying environment variables, the command issued is applied to peer0.org1. To issue commands with other peers, we will precede the environment variables in front of it. We will highlight this when we need to do so.

Then we create channel, using the *channel.tx* we created in previous section

# export CHANNEL\_NAME=mychannel

# peer channel create -o orderer.example.com:7050 -c $CHANNEL\_NAME -f ./channel-artifacts/channel.tx --tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem

Note that in each command is issued from the Orderer when channel is created.

A file *mychannel.block* is created. And the peers are joining the channel with this *mychannel.block*.

After the channel mychannel is created, we can join the peers to the channel.

For peer0.org1 (the default node)

# peer channel join -b mychannel.block

For peer0.Seller, we need to specify the environment variables.

# CORE\_PEER\_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/seller.example.com/users/Admin@seller.example.com/msp

CORE\_PEER\_ADDRESS=peer0.seller.example.com:9051

CORE\_PEER\_LOCALMSPID="SellerMSP"

CORE\_PEER\_TLS\_ROOTCERT\_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/seller.example.com/peers/peer0.seller.example.com/tls/ca.crt

peer channel join -b mychannel.block

For peer1.Buyer,

# CORE\_PEER\_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/buyer.example.com/users/Admin@buyer.example.com/msp

CORE\_PEER\_ADDRESS=peer1.buyer.example.com:8051

CORE\_PEER\_LOCALMSPID="BuyerMSP"

CORE\_PEER\_TLS\_ROOTCERT\_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/buyer.example.com/peers/peer1.buyer.example.com/tls/ca.crt

peer channel join -b mychannel.block

For peer1.Seller,

# CORE\_PEER\_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/seller.example.com/users/Admin@seller.example.com/msp

CORE\_PEER\_ADDRESS=peer1.seller.example.com:10051

CORE\_PEER\_LOCALMSPID="SellerMSP"

CORE\_PEER\_TLS\_ROOTCERT\_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/seller.example.com/peers/peer1.seller.example.com/tls/ca.crt

peer channel join -b mychannel.block

Here is how to join the four peers to the channel using *mychannel.block*.

The last step is to update the Anchor Peer on each organization. Here we use the files we created in previous section (*Org1MSPanchor.tx* and *Org2MSPanchor.tx*) and apply them to Peer0 of both Org1 and Org2.

For peer0.Buyer (the default node),

CORE\_PEER\_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/buyer.example.com/users/Admin@buyer.example.com/msp

CORE\_PEER\_ADDRESS=peer0.buyer.example.com:7051

CORE\_PEER\_LOCALMSPID="BuyerMSP"

CORE\_PEER\_TLS\_ROOTCERT\_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/buyer.example.com/peers/peer1.buyer.example.com/tls/ca.crt

peer channel update -o orderer.example.com:7050 -c $CHANNEL\_NAME -f ./channel-artifacts/BuyerMSPanchors.tx --tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem

For peer0.Seller, we need to specify environment variables as before,

# CORE\_PEER\_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/seller.example.com/users/Admin@seller.example.com/msp

CORE\_PEER\_ADDRESS=peer0.seller.example.com:9051

CORE\_PEER\_LOCALMSPID="SellerMSP"

CORE\_PEER\_TLS\_ROOTCERT\_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/seller.example.com/peers/peer0.seller.example.com/tls/ca.crt

peer channel update -o orderer.example.com:7050 -c $CHANNEL\_NAME -f ./channel-artifacts/SellerMSPanchors.tx --tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem

Step 5: Use CLI to install and instantiate the chaincode

There are two steps here.

First we install the chaincode to the peers. In our case, we first install it on both Peer0 Buyer and Peer0 Seller. The chaincode is specified in -p option in the command, and the name of chaincode is *mycc*.

On peer0.Buyer (default node),

CORE\_PEER\_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/buyer.example.com/users/Admin@buyer.example.com/msp

CORE\_PEER\_ADDRESS=peer0.buyer.example.com:7051

CORE\_PEER\_LOCALMSPID="BuyerMSP"

CORE\_PEER\_TLS\_ROOTCERT\_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/buyer.example.com/peers/peer1.buyer.example.com/tls/ca.crt

peer chaincode install -n ecommerce -v 1.0 -p github.com/chaincode/ecommerce/go/

On peer0.Seller,

#

CORE\_PEER\_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/seller.example.com/users/Admin@seller.example.com/msp

CORE\_PEER\_ADDRESS=peer0.seller.example.com:9051

CORE\_PEER\_LOCALMSPID="SellerMSP"

CORE\_PEER\_TLS\_ROOTCERT\_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/seller.example.com/peers/peer0.seller.example.com/tls/ca.crt

peer chaincode install -n ecommerce -v 1.0 -p github.com/chaincode/ecommerce/go/

After we install the chaincode on these two nodes, let’s take a look on the running containers (use another terminal).

No new containers are running. Installation chaincode does not create new containers.

The second step is to instantiate the chaincode installed from peer0.Seller. Note the environment variables we specify here.

#

CORE\_PEER\_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/seller.example.com/users/Admin@seller.example.com/msp

CORE\_PEER\_ADDRESS=peer0.seller.example.com:9051

CORE\_PEER\_LOCALMSPID="SellerMSP"

CORE\_PEER\_TLS\_ROOTCERT\_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/seller.example.com/peers/peer0.seller.example.com/tls/ca.crt

peer chaincode instantiate -o orderer.example.com:7050 --tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C $CHANNEL\_NAME -n ecommerce -v 1.0 -c '{"Args":[""]}' -P "OR ('BuyerMSP.peer','SellerMSP.peer')"

When instantiating the chaincode, we need to specify

the required arguments (-c): a JSON to initialize a with value 100, b with 200 (the required argument is defined by chaincode itself, which is not covered in this article)

the endorsing policy (-P): here for any invocation, we need both organization to endorse before the transaction is accepted. Note the AND inside the policy

If we check the containers, we see a new container is running, which is the chaincode instantiated in peer0.org2.

So after instantiating the chaincode from peer0.org2, we see one more container running, pairing the peer0.org2.

Now we query the value a from peer0.org1. No environment variables are needed as it is the default node.

Not needed for Ecommerce chaincode.

If you notice it takes some time to get the result. If we take a look on the container, we see one more container running, the chaincode for peer0.org1.

The query on peer0.org1 causes the chaincode container instantiated. Nevertheless, the result is correct, the value of a is 100.

So if we query a node which the chaincode is not running (yet), a new container is up and running, pairing that node (here it is peer0.org1).

Step 6: Use CLI to invoke the chaincode

We now invoke the chaincode (equivalent to issue a transaction) from peer0.org1.

The command to invoke the chaincode with arguments (in this example we move 10 from a to b) is,

Project:

CORE\_PEER\_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/buyer.example.com/users/Admin@buyer.example.com/msp

CORE\_PEER\_ADDRESS=peer0.buyer.example.com:7051

CORE\_PEER\_LOCALMSPID="BuyerMSP"

CORE\_PEER\_TLS\_ROOTCERT\_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/buyer.example.com/peers/peer1.buyer.example.com/tls/ca.crt

peer chaincode invoke -o orderer.example.com:7050 --tls true --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C $CHANNEL\_NAME -n ecommerce --peerAddresses peer0.buyer.example.com:7051 --tlsRootCertFiles /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/buyer.example.com/peers/peer0.buyer.example.com/tls/ca.crt --peerAddresses peer0.seller.example.com:9051 --tlsRootCertFiles /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/seller.example.com/peers/peer0.seller.example.com/tls/ca.crt -c '{"Args":["createProduct","bag","black","25","100"]}'

**Query: From Peer 0 of Buyer**

peer chaincode invoke -o orderer.example.com:7050 --tls true --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C $CHANNEL\_NAME -n ecommerce -c '{"Args":["queryAsset","Product1"]}'

Note there are many parameters passed in this command. We need to specify the peer addresses (peer0.org1 and peer0.org2) and provide the root certificate.

For sake of demonstration (and to align with what *byfn.sh* has done) we install the chaincode on peer1.org2. And then query from peer1.org2 for the latest value of a.

Install chaincode on peer1.org2,

#

Peer1:Seller

CORE\_PEER\_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/seller.example.com/users/Admin@seller.example.com/msp

CORE\_PEER\_ADDRESS=peer1.seller.example.com:10051

CORE\_PEER\_LOCALMSPID="SellerMSP"

CORE\_PEER\_TLS\_ROOTCERT\_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/seller.example.com/peers/peer1.seller.example.com/tls/ca.crt

peer chaincode install -n ecommerce -v 1.0 -p github.com/chaincode/ecommerce/go/

Query :

peer chaincode invoke -o orderer.example.com:7050 --tls true --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C $CHANNEL\_NAME -n ecommerce -c '{"Args":["queryAsset","Product1"]}'

Add one more product:

peer chaincode invoke -o orderer.example.com:7050 --tls true --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C $CHANNEL\_NAME -n ecommerce -c '{"Args":["createProduct","shoes","black","25","100"]}'

Note: Product added successfully but no output when we query

// query product:

peer chaincode invoke -o orderer.example.com:7050 --tls true --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C $CHANNEL\_NAME -n ecommerce -c '{"Args":["queryAsset","Product1"]}'

// query all Asset

peer chaincode invoke -o orderer.example.com:7050 --tls true --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C $CHANNEL\_NAME -n ecommerce -c '{"Args":["queryAllAsset"]}

Note: Product 2 added successfully but no output when we see queryallAssest

//update product

peer chaincode invoke -o orderer.example.com:7050 --tls true --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C $CHANNEL\_NAME -n ecommerce -c '{"Args":["updateProduct","dummy","Product1", "socks","white","30","100"]}'

Note: Product 1 updated successfully but no output when we see query

// order product

peer chaincode invoke -o orderer.example.com:7050 --tls true --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C $CHANNEL\_NAME -n ecommerce -c '{"Args":["createOrder","kuldeep","Product1","10"]}'

Note : how to see order product details????? And by default what will be the status of the order- ORDERED or SUBMITED or RECEIVED??????

// update order status

peer chaincode invoke -o orderer.example.com:7050 --tls true --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C $CHANNEL\_NAME -n ecommerce -c '{"Args":["updateOrderStatus","Order1","RECEIVED"]}'

Note : how to see order product details????? And by default what will be the status of the order- ORDERED or SUBMITED or RECEIVED??????

// getHistoryForRecord 1 (pid)

peer chaincode invoke -o orderer.example.com:7050 --tls true --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C $CHANNEL\_NAME -n ecommerce -c '{"Args":["getHistoryForRecord","Product1"]}'

Note: No update in the history of Product1, why????/

For upgrading chaincode:

// upgrade chaincode - upgrade instead of instantiate if version change is done

// list chaincode installed and instantiated

peer chaincode list --installed

peer chaincode invoke -o orderer.example.com:7050 --tls --cafile $ORDERER\_CA -C channel12 -n ecommerce -c '{"Args":["queryAsset","Product1"]}'

Step 7: Tear down all the setup

Remember we build this infrastructure up using docker-compose and the configuration file *docker-compose-cli.yaml*. We will tear the First Network down using similar command.

In local host, (use exit the CLI shell back to local host)

# exit

$ docker-compose -f docker-compose-cli.yaml down

Note that the docker-compose only tears down those containers specified in docker-compose-cli.yaml file. The three chaincode containers are now stopped but not removed yet. We can use command to remove everything.

$ docker rm $(docker ps -aq)

Extra:

<https://codeshare.io/5Xxv8K>

export CHANNEL\_NAME=mychannel

Echo $CHANNEL\_NAME

peer chaincode list --installed

peer chaincode list --instantiated -C mychannel

peer channel getinfo -c mychannel

peer chaincode list –installed

<https://docs.google.com/document/d/1oVxb_7BUMZ9RKXUDTtij96GyRbdfngDEWiQrisaZIAM/edit>

Endorrsment peer ???

ISSUES:

I am able to successfully execute all the commands.

But below are the issues:

1. Second product is not added in the list
2. Update to the first product is not displaying
3. Query all is not displaying all products
4. History of product is not displaying
5. Where to see order product details??