Setting up Chaincode Development Environment Using Hyperledger Fabric V1.0

we present the procedure to start fabric environment in development mode. The dev mode can be used during chaincode development and testing. In production mode, chaincodes are started and managed by the fabric itself. Whereas, in development mode, we need to start the chaincode manually on a peer. To start a development environment, we need to perform the following seven tasks:

1. Create genesis block for the orderer.
2. Start the orderer.
3. Start the peer.
4. Create channel configuration transaction and create a channel on orderer.
5. Join the peer to created channel.
6. Compile the chaincode and start executing.
7. Install and Instantiate the chaincode on the peer and invoke the chaincode

Note: Run all commands for above tasks from *$GOPATH/src/github.com/hyperledger/fabric/* unless specified otherwise.

First : Install prerequisites

Before we begin, if you haven't already done so, you may wish to check that you have all the Prerequisites installed on the platform(s) on which you'll be developing blockchain applications and/or operating Hyperledger Fabric.

Second:Unzip the file called localOptim

Start the network

We want to go through the commands manually in order to expose the syntax and functionality of each call.

run the following command:

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

Environment variables

For the following CLI commands against peer0.org1.example.com to work, we need to preface our commands with the four environment variables given below. These variables for peer0.org1.example.com are baked into the CLI container, therefore we can operate without passing them. **HOWEVER**, if you want to send calls to other peers or the orderer, then you will need to provide these values accordingly. Inspect the docker-compose-base.yaml for the specific paths:

Environment variables for PEER0
##CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
##CORE_PEER_ADDRESS=peer0.org1.example.com:7051
##CORE_PEER_LOCALMSPID="Org1MSP"

##CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt

Create & Join Channel:

We will enter the CLI container using the docker exec command:

##docker exec -it cli bash
If successful you should see the following:

root@0d78bb69300d:/opt/gopath/src/github.com/hyperledger/fabric/peer#

Next, we are going to pass in the generated channel configuration transaction artifact that we created in the Create a Channel Configuration Transaction section (we called it channal.tx) to the orderer as part of the create channel request.

##export CHANNEL_NAME=ch1

the channel.tx file is mounted in the channel-artifacts directory within your CLI container
as a result, we pass the full path for the file
we also pass the path for the orderer ca-cert in order to verify the TLS handshake
be sure to export or replace the $CHANNEL_NAME variable appropriately

##peer channel create -o orderer.example.com:7050 -c ch1 -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

This command returns a genesis block - <ch1.block> - which we will use to join the channel. It contains the configuration information specified in channel.tx If you have not made any modifications to the default channel name, then the command will return you a proto titled ch1.block.

Now let’s join peer0.org1.example.com to the channel.

By default, this joins ``peer0.org1.example.com`` only
the <channel-ID.block> was returned by the previous command
if you have not modified the channel name, you will join with mychannel.block
if you have created a different channel name, then pass in the appropriately named block

##peer channel join -b mychannel.block

You can make other peers join the channel as necessary by making appropriate changes in the four environment variables we used in the Environment variables section, above.

Install & Instantiate Chaincode

First, install the chaincode onto one of the three peer nodes. These commands place the specified source code flavor onto our peer's filesystem.

##this installs the chaincode

##peer chaincode install -n mycc -v 1.0 -p github.com/chaincode/chain/go/

Next, instantiate the chaincode on the channel. This will initialize the chaincode on the channel, set the endorsement policy for the chaincode, and launch a chaincode container for the targeted peer. Take note of the -P argument. This is our policy where we specify the required level of endorsement for a transaction against this chaincode to be validated.

be sure to replace the $CHANNEL_NAME environment variable if you have not exported it
if you did not install your chaincode with a name of mycc, then modify that argument as well

##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 mycc -v 1.0 -c '{"Args":["init"]}' -P

Now issue an invoke .

be sure to set the -C and -n flags appropriately

##peer chaincode invoke -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 ch1 -n mycc -c '{"Args":["writeX_1","x_1","25"]}'

##peer chaincode invoke -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 ch1 -n mycc -c '{"Args":["writeX_2","x_2","-20"]}'

##peer chaincode invoke -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 ch1 -n mycc -c '{"Args":["writeX_3","x_3","25"]}'

Now run the following commands to get Newlamda

##peer chaincode invoke -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 ch1 -n mycc -c '{"Args":["ComputLamb"]}'

Finally, query Lambda

peer chaincode query -n mycc -c '{"Args":["query","Lambda"]}' -C myc

Troubleshooting

- Always start your network fresh. Use the following command to remove artifacts, crypto, containers and chaincode images:

  ./byfn.sh -m down

  Note:

  You **will** see errors if you do not remove old containers and images.

- If you see Docker errors, first check your docker version (Prerequisites), and then try restarting your Docker process. Problems with Docker are oftentimes not immediately recognizable. For example, you may see errors resulting from an inability to access crypto material mounted within a container.

  If they persist remove your images and start from scratch:

  ##docker rm -f $(docker ps -aq)
  ##docker rmi -f $(docker images -q)