Lab 2. Running our first network.

Prequisites: Complete Lab 1.

Now that we have installed our first network, we can run it.

Step 1. Start the network

If the network is down, start it again with the following:

```
$./byfn.sh -m up
```

This brings up the four peer nodes and the orderer node.

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.exampl
e.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/fab
ric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.ex
ample.com/tls/ca.crt
```

Now let's start our docker container that will let us run command line interface commands against peer0.

Run the following command:

```
$ docker exec -it cli bash
```

A successful completion should see a new prompt inside the docker container that looks something like this:

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

Next, we will create the channel on the peer by passing in the channel information that we created earlier.

```
$ export CHANNEL_NAME=mychannel

# 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 replace the $CHANNEL_NAME variable appropriately

$ peer channel create -o orderer.example.com:7050 -c $CHANNEL_NAME -f
./channel-artifacts/channel.tx --tls $CORE_PEER_TLS_ENABLED --cafile
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrgan
izations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca
.example.com-cert.pem
```

This command returns a genesis block - <channel-ID.block> - which we will use to join the channel. It contains the configuration information specified in channel.tx. In our case, the genesis block is in the file mychannel.block. Doing an 'ls' of the container should list it.

Now, join the peer node to the channel with the peer channel join command:

```
# By default, this joins ``peer0.org1.example.com`` only
# the <channel-ID.block> was returned by the previous command
$ peer channel join -b <channel-ID.block>
```

Note that running a peer channel list command will list all of the channels that the node is joined to.

Now let's install and instantiate some chaincode on the peer node.

Run the following command on the peer node:

```
peer chaincode install -n mycc -v 1.0 -p
github.com/hyperledger/fabric/examples/chaincode/go/chaincode_example02
```

This command will install the example, which we'll call mycc onto the peer node from the supplied github repository.

Now let's instantiate the chaincode on the node.

```
# be sure to replace the $CHANNEL_NAME environment variable
# if you did not install your chaincode with a name of mycc, then modify that argu

peer chaincode instantiate -o orderer.example.com:7050 --tls $CORE_PEER_TLS_ENABLE
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/exa
cert.pem -C $CHANNEL_NAME -n mycc -v 1.0 -c '{"Args":["init","a", "100", "b","200"
```

In the command above you'll notice that we specify our policy as -P "OR ('Org0MSP.member','Org1MSP.member')". This means that we need "endorsement" from a peer belonging to Org1 OR Org2 (i.e. only one endorsement). If we changed the syntax to AND then we would need two endorsements.

Now that we have instantiated the chaincode, we can run queries on it. Let's query the value of the 'a' parameter that was passed to the chaincode.

```
# be sure to set the -C and -n flags appropriately
peer chaincode query -C $CHANNEL_NAME -n mycc -c
'{"Args":["query","a"]}'
```

The output should be '100'.

We can also invoke the script directly using the invoke command. In this example we'll move 10 units from the variable a to the variable b.

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

peer chaincode invoke -o orderer.example.com:7050  --tls
$CORE_PEER_TLS_ENABLED --cafile
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrgan
izations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca
.example.com-cert.pem  -C $CHANNEL_NAME -n mycc -c
'{"Args":["invoke","a","b","10"]}'
```

Now run the query on the 'a' variable again. The resultant value should be 90.