**Installing Hyperledger-Fabric using docker compose on ec2**

# 

[Install prerequisites](#_d55dorbwglok)

[Install Docker and Docker Compose](#_y314mqbafnb)

[Install Fabric](#_pipvcf5c0s6y)

[Configure Go PATH](#_bdzicifg0kyi)

[Crypto Generator](#_bau1euhe685u)

[How does it work?](#_816z3ttd1b45)

[Configuration Transaction Generator](#_92txkv7wmjbs)

[Configtxgen Tool](#_yfxy2fxv02pq)

[How does it work?](#_rpkvl95o6ay6)

[Start Fabric Network](#_j4m3fp70sggr)

[Start Network](#_ddfmxrvr7e3y)

[Verify the containers](#_s5gpvqdpjwyt)

[Create & Join Channel](#_loxoar3st8n0)

[Create Channel](#_ww0uktmow7az)

[Join peers in to Channel](#_ardfw3yrrc2o)

[Update the Anchor peers](#_pzmjj42y4d3a)

[Install and define a chain code](#_ape8eykxgmrf)

[Instantiate the chain code](#_5zvpx5bhvlnj)

[Instantiate](#_s9awfahnfug3)

[Validate chaincode instantiation](#_hpi64wanaib6)

[Invoke a new transaction](#_xt4ovrc5f4x8)

[Transaction](#_n4t5to23e15h)

[Fabric Java SDK](#_y1k0l9or2vlz)

[Description](#_bf0y357afo4p)

[Thoughts](#_qq27fs61ksz4)

# Install prerequisites

* Install cURL
* Go Lang (yum install go -y)

# Install Docker and Docker Compose

$curl -L "https://github.com/docker/compose/releases/download/1.24.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

$chmod +x /usr/local/bin/docker-compose

# 

# Install Fabric

curl -sSL http://bit.ly/2ysbOFE | bash -s

# Configure Go PATH

$cd fabric-samples/first-network

$export PATH=${PWD}/../bin:${PWD}:$PATH

$export FABRIC\_CFG\_PATH=${PWD}

# Crypto Generator

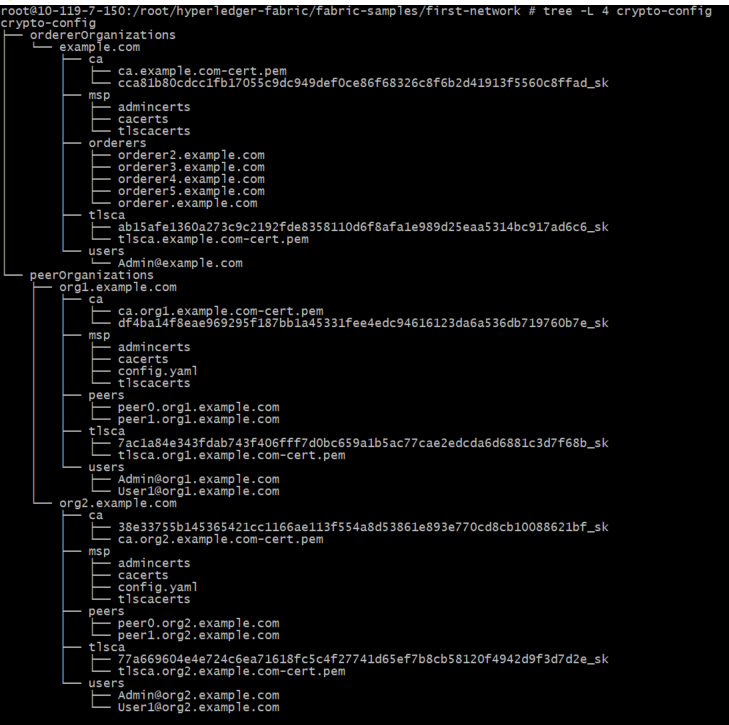
Fabric has **Cryptogen** tool to generate the cryptographic material (x509 certs and signing keys) for our various network entities. These certificates are representative of identities, and they allow for sign/verify authentication to take place as our entities communicate and transact.

## How does it work?

* Cryptogen consumes a file — ***crypto-config.yaml*** — that contains the network topology and allows us to generate a set of certificates and keys for both the Organizations and the components that belong to those Organizations.
* Each Organization is provisioned a unique root certificate (ca-cert) that binds specific components (peers and orderers) to that Org. By assigning each Organization a unique CA certificate, we are mimicking a typical network where a participating Member would use its own Certificate Authority. Transactions and communications within Hyperledger Fabric are signed by an entity’s private key (keystore), and then verified by means of a public key (signcerts).
* You will notice a count variable within this file. We use this to specify the number of peers per Organization; in our case there are two peers per Org. We won’t delve into the minutiae of x.509 certificates and public key infrastructure right now. If you’re interested, you can peruse these topics on your own time.
* After we run the cryptogen tool, the generated certificates and keys will be saved to a folder titled crypto-config. Note that the crypto-config.yaml file lists five orderers as being tied to the orderer organization. While the cryptogen tool will create certificates for all five of these orderers, unless the Raft or Kafka ordering services are being used, only one of these orderers will be used in a Solo ordering service implementation and be used to create the system channel and mychannel.

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

The cryptogen tool generates a **crypto-config** folder in the current directory based on the **crypto-config.yaml** file. The folder structure is shown as follows.



# Configuration Transaction Generator

## Configtxgen Tool

The ***configtxgen*** tool is used to create four configuration artifacts:

1. orderer genesis block
2. channel configuration transaction
3. and two anchor peer transactions - one for each Peer Org

**Orderer genesis block**

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

**Channel configuration transaction file**

configtxgen -profile TwoOrgsChannel -outputCreateChannelTx ./channel-artifacts/channel.tx -channelID mychannel

**Anchor peer for Org1 of the channel**

configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate ./channel-artifacts/Org1MSPanchors.tx -channelID mychannel -asOrg Org1MSP

**Anchor peer for Org2 of the channel**

configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate ./channel-artifacts/Org2MSPanchors.tx -channelID mychannel -asOrg Org2MSP

## 

## How does it work?

Configtxgen consumes a file - *configtx.yaml* - that contains the definitions for the sample network. There are three members - one Orderer Org (OrdererOrg) and two Peer Orgs (Org1 & Org2) each managing and maintaining two peer nodes. This file also specifies a consortium - ***SampleConsortium*** - consisting of our two Peer Orgs.

# 

# Start Fabric Network

## Start Network

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

## Verify the containers

*docker ps -a*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CONTAINER ID** | **IMAGE** | **COMMAND** | **PORTS** | **NAMES** |
| fb63fb34f079 | hyperledger/fabric-tools:latest | "/bin/bash" |  | cli |
| aae267f8fcc4 | hyperledger/fabric-peer:latest | "peer node start" | 0.0.0.0:7051->7051/tcp | peer0.org1.example.com |
| afbf33c3d8b1 | hyperledger/fabric-orderer:latest | "orderer" | 0.0.0.0:7050->7050/tcp | orderer.example.com |
| 8f32e81852e4 | hyperledger/fabric-orderer:latest | "peer node start" | 0.0.0.0:9051->9051/tcp | peer0.org2.example.com |
| 944907c67158 | hyperledger/fabric-orderer:latest | "peer node start" | 0.0.0.0:10051->10051/t | peer1.org2.example.com |
| 158e9702cde7 | hyperledger/fabric-orderer:latest | "peer node start" | 0.0.0.0:8051->8051/tcp | peer1.org1.example.com |

# 

# 

# 

# 

# 

# Create & Join Channel

Create a channel, add peer nodes to the channel, and configure an anchor point for each organization

## Create Channel

Enter into the CLI container using the following command

$docker exec -it cli bash

*peer channel create -o orderer.example.com:7050 -c mychannel -f ./channel-artifacts/channel.tx --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*

**Details:**

-o specifies the orderer node orderer.example.com:7050. In the docker-compose-base.yaml file, port 7050 is specified for the orderer node.

-c mychannel -f ./channel-artifacts/channel.tx: specifies the channel name and uses the channel.tx configuration file that has been generated to initialize the channel

--tls true: indicates that TLS is used for encrypted transmission on the network.

--cafile: specifies the CA certificate path

## Join peers in to Channel

**Peer0/Org1**

$peer channel join -b mychannel.block

**Peer1/Org1**

$export CORE\_PEER\_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp

$export CORE\_PEER\_ADDRESS=peer1.org1.example.com:8051

export CORE\_PEER\_LOCALMSPID="Org1MSP"

$export CORE\_PEER\_TLS\_ROOTCERT\_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer1.org1.example.com/tls/ca.crt

$ peer channel join -b mychannel.block

**Peer0/Org2**

$export CORE\_PEER\_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp

$export CORE\_PEER\_ADDRESS=peer0.org2.example.com:9051

$export CORE\_PEER\_LOCALMSPID="Org2MSP"

$export CORE\_PEER\_TLS\_ROOTCERT\_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt

$ peer channel join -b mychannel.block

**Peer1/Org2**

$export CORE\_PEER\_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp

$export CORE\_PEER\_ADDRESS=peer1.org2.example.com:10051

$export CORE\_PEER\_LOCALMSPID="Org2MSP"

$export CORE\_PEER\_TLS\_ROOTCERT\_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/peers/peer1.org2.example.com/tls/ca.crt

$ peer channel join -b mychannel.block

# Update the Anchor peers

Peer0 as anchor peer in both the organizations

**Anchor Point for Org1, PEER-0**

$ export CORE\_PEER\_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp

$ export CORE\_PEER\_ADDRESS=peer0.org1.example.com:7051

$ export CORE\_PEER\_LOCALMSPID="Org1MSP"

$ export 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

$ peer channel update -o orderer.example.com:7050 -c mychannel -f ./channel-artifacts/Org1MSPanchors.tx --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

**Anchor Point for Org2, PEER-0**

export CORE\_PEER\_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp

export CORE\_PEER\_ADDRESS=peer0.org2.example.com:9051

export CORE\_PEER\_LOCALMSPID="Org2MSP"

export CORE\_PEER\_TLS\_ROOTCERT\_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt

$peer channel update -o orderer.example.com:7050 -c mychannel -f ./channel-artifacts/Org2MSPanchors.tx --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

# 

# Install and define a chain code

Go lang in this example

**Peer0/Org1**

CORE\_PEER\_MSPCONFIGPATH=/opt/gopath/src/[github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp](http://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 peer chaincode install -n mycc -v 1.0 -p github.com/chaincode/chaincode\_example02/go/

**Peer1/Org1**

CORE\_PEER\_MSPCONFIGPATH=/opt/gopath/src/[github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp](http://github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp) CORE\_PEER\_ADDRESS=peer1.org1.example.com:**8051** CORE\_PEER\_LOCALMSPID="Org1MSP" CORE\_PEER\_TLS\_ROOTCERT\_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer1.org1.example.com/tls/ca.crt peer chaincode install -n mycc -v 1.0 -p github.com/chaincode/chaincode\_example02/go/

**Peer0/Org2**

CORE\_PEER\_MSPCONFIGPATH=/opt/gopath/src/[github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp](http://github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp) CORE\_PEER\_ADDRESS=peer0.org2.example.com:**9051** CORE\_PEER\_LOCALMSPID="Org2MSP" CORE\_PEER\_TLS\_ROOTCERT\_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt peer chaincode install -n mycc -v 1.0 -p github.com/chaincode/chaincode\_example02/go/

**Peer1/Org2**

CORE\_PEER\_MSPCONFIGPATH=/opt/gopath/src/[github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp](http://github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp) CORE\_PEER\_ADDRESS=peer1.org2.example.com:**10051** CORE\_PEER\_LOCALMSPID="Org2MSP" CORE\_PEER\_TLS\_ROOTCERT\_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/peers/peer1.org2.example.com/tls/ca.crt peer chaincode install -n mycc -v 1.0 -p github.com/chaincode/chaincode\_example02/go/

# Instantiate the chain code

## Instantiate

$peer chaincode instantiate -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 mychannel -n mycc -l golang -v 1.0 -c '{"Args":["init","a","100","b","200"]}' -P 'OR ('\''Org1MSP.peer'\'','\''Org2MSP.peer'\'')'

**Check the logs for peer0**

$docker logs --tail 50 peer0.org1.example.com

**Logs output:**

2019-06-28 19:18:05.470 UTC [endorser] **callChaincode** -> INFO 061 [mychannel][a30e4471] Entry chaincode: name:"lscc"

2019-06-28 19:18:05.478 UTC [chaincode.platform.golang] GenerateDockerBuild -> INFO 062 building chaincode with ldflagsOpt: '-ldflags "-linkmode external -extldflags '-static'"'

2019-06-28 19:18:26.949 UTC [endorser] callChaincode -> INFO 063 [mychannel][a30e4471] Exit chaincode: name:"lscc" (21479ms)

2019-06-28 19:18:26.949 UTC [comm.grpc.server] 1 -> INFO 064 unary call completed grpc.service=protos.Endorser grpc.method=ProcessProposal grpc.peer\_address=172.19.0.7:55988 grpc.code=OK grpc.call\_duration=21.480520694s

2019-06-28 19:18:28.957 UTC [gossip.privdata] StoreBlock -> INFO 065 [mychannel] Received block [3] from buffer

2019-06-28 19:18:28.961 UTC [**committer.txvalidator] Validate** -> INFO 066 [mychannel] Validated block [3] in 3ms

2019-06-28 19:18:28.961 UTC [cceventmgmt] **HandleStateUpdates** -> INFO 067 Channel [mychannel]: Handling deploy or update of chaincode [mycc]

2019-06-28 19:18:28.970 UTC [kvledger] **CommitWithPvtData** -> INFO 068 [mychannel] Committed block [3] with 1 transaction(s) in 8ms (state\_validation=0ms block\_commit=5ms state\_commit=1ms)

## 

## Validate chaincode instantiation

$peer chaincode query -C mychannel -n mycc -c '{"Args":["query","a"]}'

Output:100

$peer chaincode query -C mychannel -n mycc -c '{"Args":["query","b"]}'

Output: 200

# Invoke a new transaction

## Transaction

Transfer $10 from variable “a” -> variable “b”

$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 mychannel -n mycc -c '{"Args":["invoke","a","b","10"]}'

**Check Logs**

$docker logs --tail 200 dev-peer0.org1.example.com-mycc-1.0

**Output:**

ex02 Init

Aval = 100, Bval = 200

ex02 Invoke

Query Response:{"Name":"a","Amount":"100"}

ex02 Invoke

Query Response:{"Name":"b","Amount":"200"}

ex02 Invoke

Aval = 90, Bval = 210

ex02 Invoke

Query Response:{"Name":"a","Amount":"90"}

ex02 Invoke

Query Response:{"Name":"b","Amount":"210"}

**ChainCode - Java - Source:**

**This is actual contract/chaincode installed by the orderer**

1. **ChaincodeBase implements ChaincodeStub - this interface has all the methods to handle the chain code inlcuding invoking the chiancode, getState, putState, delState, getQueryResult, getPrivateData, getEvent/ChaincodeEvent etc...**

**public class SimpleChaincode extends org.hyperledger.fabric.shim.ChaincodeBase**

**@Override**

**public Response init(ChaincodeStub stub) {**

**// business logic initialization happens here**

**}**

**@Override**

**public Response invoke(ChaincodeStub stub) {**

**// this method must handle when new transaction invoked by the peer(s).**

**}**

# Fabric Java SDK

## Description

The SDK helps facilitate Java applications to manage the lifecycle of Hyperledger channels and user chaincode. The SDK also provides a means to execute user chaincode, query blocks and transactions on the channel, and monitor events on the channel.

<https://github.com/hyperledger/fabric-sdk-java>

## Thoughts

Research on the SDK and see if Verizon needs a customized version of this SDK. This will help to manage the life cycle of the hyperledger channels and chaincode.

# Using Private Data in Fabric

## Build Collection Definition

A collection definition is composed of the following properties:

***name***: Name of the collection.

***policy***: Defines the organization peers allowed to persist the collection data.

***requiredPeerCount***: Number of peers required to disseminate the private data as a condition of the endorsement of the chaincode

***maxPeerCount***: For data redundancy purposes, the number of other peers that the current endorsing peer will attempt to distribute the data to. If an endorsing peer goes down, these other peers are available at commit time if there are requests to pull the private data.

***blockToLive***: For very sensitive information such as pricing or personal information, this value represents how long the data should live on the private database in terms of blocks. The data will live for this specified number of blocks on the private database and after that it will get purged, making this data obsolete from the network. To keep private data indefinitely, that is, to never purge private data, set the blockToLive property to 0.

***memberOnlyRead***: a value of true indicates that peers automatically enforce that only clients belonging to one of the collection member organizations are allowed read access to private data.

**Example**

**// collections\_config.json**

[ {

"name": "collectionMarbles", "policy": "OR('Org1MSP.member', 'Org2MSP.member')", "requiredPeerCount": 0,

"maxPeerCount": 3, "blockToLive":1000000, "memberOnlyRead": true

},

{

"name": "collectionMarblePrivateDetails", "policy": "OR('Org1MSP.member')", "requiredPeerCount": 0, "maxPeerCount": 3, "blockToLive":3, "memberOnlyRead": true

}

]

## Read and Write private data using chaincode APIs

### Use Case

|  |
| --- |
| *// Peers in Org1 and Org2 will have this private data in a side database*  **type** marble **struct** {  ObjectType **string** `json:"docType"`  Name **string** `json:"name"`  Color **string** `json:"color"`  Size **int** `json:"size"`  Owner **string** `json:"owner"`  }  *// Only peers in Org1 will have this private data in a side database*  **type** marblePrivateDetails **struct** {  ObjectType **string** `json:"docType"`  Name **string** `json:"name"`  Price **int** `json:"price"`  } |

Specifically access to the private data will be restricted as follows:

name, color, size, and owner will be visible to all members of the channel (Org1 and Org2)

price only visible to members of Org1

Thus two different sets of private data are defined in the marbles private data sample. The mapping of this data to the collection policy which restricts its access is controlled by chaincode APIs. Specifically, reading and writing private data using a collection definition is performed by calling GetPrivateData() and PutPrivateData().

### 

### Reading Collection Data

Use the chaincode API GetPrivateData() to query private data in the database. GetPrivateData() takes two arguments, the collection name and the data key.

### 

### Writing private Data

Use the chaincode API PutPrivateData() to store the private data into the private database. The API also requires the name of the collection.

# Install Private Data Chaincode Sample

Install the chaincode to the existing BYFN network.

Enter into the CLI container using the following command

$docker exec -it cli bash

**Install chaincode**

$peer chaincode install -n **marbles\_pdc** -v 1.0 -p github.com/chaincode/marbles02\_private/go/

**marbles\_pdc** is the name of chaincode installed

You should see following

2019-07-05 19:33:48.419 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 001 Using default escc

2019-07-05 19:33:48.419 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 002 Using default vscc

2019-07-05 19:33:48.647 UTC [chaincodeCmd] install -> INFO 003 Installed remotely response:<status:200 payload:"OK" >

Run following command to list all the chaincodes installed in this peer

$peer chaincode list --installed

* Use the CLI to switch the active peer to the second peer in Org1 and install the chaincode. Copy and paste the following entire block of commands into the CLI container and run them.

|  |
| --- |
| **$export CORE\_PEER\_ADDRESS=peer1.org1.example.com:8051**  **peer chaincode install -n marbles\_pdc -v 1.0 -p github.com/chaincode/marbles02\_private/go/** |

* Use the CLI to switch to Org2. Copy and paste the following block of commands as a group into the peer container and run them all at once.

|  |
| --- |
| **$export CORE\_PEER\_LOCALMSPID=Org2MSP**  **$export PEER0\_ORG2\_CA=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt**  **$export CORE\_PEER\_TLS\_ROOTCERT\_FILE=$PEER0\_ORG2\_CA**  **$export $CORE\_PEER\_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp** |

* Switch the active peer to the first peer in Org2 and install the chaincode

|  |
| --- |
| **$export CORE\_PEER\_ADDRESS=peer0.org2.example.com:9051**  **$peer chaincode install -n marbles\_pdc -v 1.0 -p github.com/chaincode/marbles02\_private/go/** |

* Switch the active peer to the second peer in org2 and install the chaincode

|  |
| --- |
| **$export CORE\_PEER\_ADDRESS=peer1.org2.example.com:10051**  **$peer chaincode install -n marbles\_pdc -v 1.0 -p github.com/chaincode/marbles02\_private/go/** |

# 

# Instantiate Chaincode on the channel

Use the [peer chaincode instantiate](http://hyperledger-fabric.readthedocs.io/en/master/commands/peerchaincode.html?%20chaincode%20instantiate#peer-chaincode-instantiate) command to instantiate the marbles chaincode on a channel. To configure the chaincode collections on the channel, specify the flag --collections-config along with the name of the collections JSON file, collections\_config.json in our example.

|  |
| --- |
| **$export ORDERER\_CA=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem**  **$peer chaincode instantiate -o orderer.example.com:7050 --tls --cafile $ORDERER\_CA -C mychannel -n marblesp -v 1.0 -c '{"Args":["init"]}' -P "OR('Org1MSP.member','Org2MSP.member')" --collections-config $GOPATH/src/github.com/chaincode/marbles02\_private/collections\_config.json** |

# Store private data

* Acting as a member of Org1, who is authorized to transact with all of the private data in the marbles private data sample, switch back to an Org1 peer and submit a request to add a marble.

|  |
| --- |
| $export CORE\_PEER\_ADDRESS=peer0.org1.example.com:7051  $export CORE\_PEER\_LOCALMSPID=Org1MSP  $export 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  $export CORE\_PEER\_MSPCONFIGPATH=/opt/gopath/src/[github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp](http://github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp)  $export PEER0\_ORG1\_CA=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt |

* Invoke the marbles initMarble function which creates a marble with private data — name marble1 owned by tom with a blue color, size 35 and price of 99.

|  |
| --- |
| $export MARBLE=$(echo -n "{\"name\":\"marble1\",\"color\":\"blue\",\"size\":35,\"owner\":\"tom\",\"price\":99}" | base64 | tr -d \\n)  $peer chaincode invoke -o orderer.example.com:7050 --tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C mychannel -n marblesp -c '{"Args":["initMarble"]}' --transient "{\"marble\":\"$MARBLE\"}" |

# Query the private data as an authorized peer

Our collection definition allows all members of Org1 and Org2 to have the name, color, size, owner private data in their side database, but only peers in Org1 can have the price private data in their side database. As an authorized peer in Org1, we will query both sets of private data.

|  |
| --- |
| $peer chaincode query **-**C mychannel **-**n marbles\_pdc **-**c '{"Args":["readMarble","marble1"]}' |

You should see the following result:

|  |
| --- |
| {"color":"blue","docType":"marble","name":"marble1","owner":"tom","size":35} |

## Query for the price private data of marble1 as a member of Org1

|  |
| --- |
| $peer chaincode query -C mychannel -n marbles\_pdc -c '{"Args":["readMarblePrivateDetails","marble1"]}' |

You should see the following result:

|  |
| --- |
| {"docType":"marblePrivateDetails","name":"marble1","price":99} |

# Query the private data as an unauthorized peer

Now we will switch to a member of Org2 which has the marbles private data name, color, size, owner in its side database, but does not have the marbles price private data in its side database. We will query for both sets of private data.

## Switch to a peer in Org2

From inside the docker container, run the following commands to switch to the peer which is unauthorized to access the marbles price private data.

|  |
| --- |
| $export CORE\_PEER\_ADDRESS=peer0.org2.example.com:9051  $export CORE\_PEER\_LOCALMSPID=Org2MSP  $export PEER0\_ORG2\_CA=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt  $export CORE\_PEER\_TLS\_ROOTCERT\_FILE=$PEER0\_ORG2\_CA  $export CORE\_PEER\_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp |

## Query private data Org2 is authorized to

Peers in Org2 should have the first set of marbles private data (name, color, size and owner) in their side database and can access it using the readMarble() function which is called with the collectionMarbles argument.

|  |
| --- |
| $peer chaincode query -C mychannel -n marblesp -c '{"Args":["readMarble","marble1"]}' |

You should see something similar to the following result:

|  |
| --- |
| {"docType":"marble","name":"marble1","color":"blue","size":35,"owner":"tom"} |

## Query private data Org2 is not authorized to

Peers in Org2 do not have the marbles price private data in their side database. When they try to query for this data, they get back a hash of the key matching the public state but will not have the private state.

|  |
| --- |
| peer chaincode query -C mychannel -n marblesp -c '{"Args":["readMarblePrivateDetails","marble1"]}' |

Error: endorsement failure during query. response: status:500 message:"{\"Error\":\"Failed to get private details for marble1: GET\_STATE failed: transaction ID: b3f49a387b4bfa06f74a392e7dcd81a8aa7575f9369ae7511188a1063098a218: tx creator does not have read access permission on privatedata in chaincodeName:marbles\_pdc collectionName: collectionMarblePrivateDetails\"}"