

< 하이퍼레저 페브릭-3 >

2018. 11. 20(화)

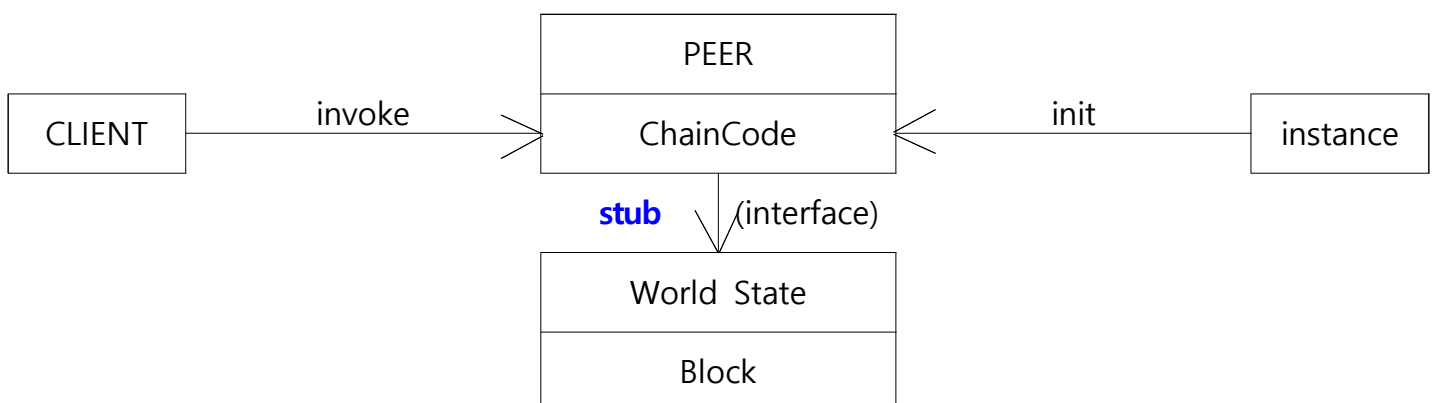
1. go lang의 문법

```
package main

import (
    "fmt"
    "strconv"
    "github.com/hyperledger/fabric/core/chaincode/shim"
    "github.com/hyperledger/fabric/protos/peer"
)

// SimpleAsset implements a simple chaincode to manage an asset
type SimpleAsset struct {
}
```

- package main과 import는 필수적으로 써줘야 하는 것(just 문법)
- shim은 연결해주는 interface(약속) 역할 => client와 chaincode를 연결해주는 역할
- peer는 원장에 최종 작성을 하기 위해 작성해주는 것
- type SimpleAsset struct는 class의 개념으로 생각(일종의 상자 같은 개념)
: 내가 필요한 자료형을 만드는 것(실습 간 예제에서는 내용을 채우지 않았지만, marbles에서 쓰이는 원형 같은 것들이 이 안에 쓰임)
- **\$ func (t *SimpleAsset) Init(stub shim.ChaincodeStubInterface) peer.Response**
python의 self같은 역할(SimpleAsset 안에서 쓰겠다) 반환해주는 함수(Response로 return)
- stub은 대리 역할을 해주는 agent라고 생각
: client가 invoke를 통해 CC를 호출하면, CC가 블록(World State)에 접근하기 위해 쓰이는 매개체
: instance를 만들기 위해 init을 호출할 때도 마찬가지로(문을 여는 열쇠 같은 개념으로 생각)
: 밖에서(client) 쓰인 변수를 안으로(world state) 가져와 쓰기 위해서 거쳐야 할 interface!!



2. ChainCode 수정 후 버전 업그레이드

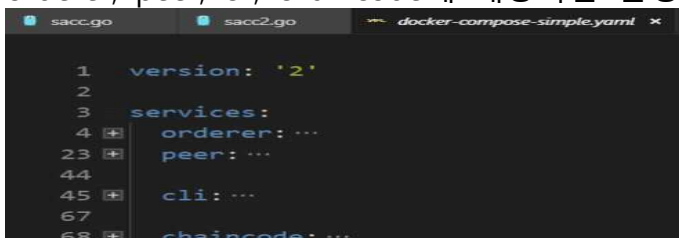
- fabric-samples에서는 기본으로 제공해주는 개발자용 네트워크가 있음(chaincode-docker-devmode)

```
guru@guru:~/fabric-samples$ ll
total 96
drwxrwxr-x 14 guru guru 4096 Nov 15 13:20 ./
drwxr-xr-x 9 guru guru 4096 Nov 15 14:39 ../
drwxrwxr-x 5 guru guru 4096 Nov 14 13:42 balance-transfer/
drwxrwxr-x 4 guru guru 4096 Nov 14 13:42 basic-network/
drwxrwxr-x 2 guru guru 4096 Oct 10 23:55 bin/
drwxrwxr-x 10 guru guru 4096 Nov 19 14:54 chaincode/
drwxrwxr-x 4 guru guru 4096 Nov 19 14:32 chaincode-docker-devmode/
-rw-rw-r-- 1 guru guru 597 Nov 14 13:42 CODE_OF_CONDUCT.md
drwxrwxr-x 2 guru guru 4096 Oct 10 23:41 config/
-rw-rw-r-- 1 guru guru 961 Nov 14 13:42 CONTRIBUTING.md
drwxrwxr-x 4 guru guru 4096 Nov 15 14:04 fabcar/
drwxrwxr-x 3 guru guru 4096 Nov 14 13:42 fabric-ca/
drwxrwxr-x 8 guru guru 4096 Nov 16 16:24 first-network/
drwxrwxr-x 8 guru guru 4096 Nov 14 13:42 .git/
-rw-rw-r-- 1 guru guru 130 Nov 14 13:42 .gitignore
-rw-rw-r-- 1 guru guru 109 Nov 14 13:42 .gitreview
drwxrwxr-x 4 guru guru 4096 Nov 14 13:42 high-throughput/
-rw-rw-r-- 1 guru guru 3193 Nov 14 13:42 Jenkinsfile
-rw-rw-r-- 1 guru guru 11358 Nov 14 13:42 LICENSE
-rw-rw-r-- 1 guru guru 470 Nov 14 13:42 MAINTAINERS.md
-rw-rw-r-- 1 guru guru 1230 Nov 14 13:42 README.md
drwxrwxr-x 3 guru guru 4096 Nov 14 13:42 scripts/
```

- 이 안에 들어가 보면 myc.block, tx, orderer 등 기본 설정해줘야 할 파일들이 이미 생성되어 들어가 있음.

```
guru@guru:~/fabric-samples/chaincode-docker-devmode$ ll
total 60
drwxrwxr-x 4 guru guru 4096 Nov 19 14:32 ./
drwxrwxr-x 14 guru guru 4096 Nov 15 13:20 ../
drwxr-xr-x 2 root root 4096 Nov 19 14:32 chaincode/
-rw-rw-r-- 1 guru guru 2702 Nov 14 13:42 docker-compose-simple.yaml
-rw-rw-r-- 1 guru guru 83 Nov 14 13:42 .gitignore
drwxrwxr-x 8 guru guru 4096 Nov 14 13:42 msp/
-rw-r--r-- 1 root root 10876 Nov 19 14:32 myc.block
-rw-rw-r-- 1 guru guru 274 Nov 14 13:42 myc.tx
-rw-rw-r-- 1 guru guru 7902 Nov 14 13:42 orderer.block
-rw-rw-r-- 1 guru guru 5270 Nov 14 13:42 README.rst
-rwxrwxr-x 1 guru guru 1019 Nov 14 13:42 script.sh*
```

- \$ rmate -p 52698 docker-compose-simple.yaml로 띄워서 보면
orderer, peer, cli, chaincode에 해당하는 환경변수 값, 포트번호 등이 다 작성되어 있음



```
sacc.go sacc2.go docker-compose-simple.yaml x
1 version: '2'
2
3 services:
4   orderer: ...
23   peer: ...
44
45   cli: ...
67
68   chaincode: ...
```

- \$ **docker-compose -f docker-compose-simple.yaml up -d**로 네트워크 기동(up)
- \$ **docker exec -it chaincode bash**로 체인코드 컨테이너로 이동!
- \$ **CORE_PEER_ADDRESS=peer:7052 CORE_CHAINCODE_ID_NAME=mycc:2 ./sacc2**
: 네트워크 기본값들을 설정(주소는 7052포트를 이용하고, 체인코드 이름은 mycc, 버전은 2, ./sacc2파일로 실행하겠다)
라고 엔터를 치면 'starting up'으로 시작이 됨

```
root@a9f8766e7797:/opt/gopath/src/chaincode/sacc2# CORE_PEER_ADDRESS=peer:7052 CORE_CHAINCODE_ID_NAME=mycc:2 ./sacc2
2018-11-20 01:44:00.114 UTC [shim] setupChaincodeLogging -> INFO 001 Chaincode log level not provided; defaulting to: INFO
2018-11-20 01:44:00.117 UTC [shim] setupChaincodeLogging -> INFO 002 Chaincode (build level: ) starting up ...
```

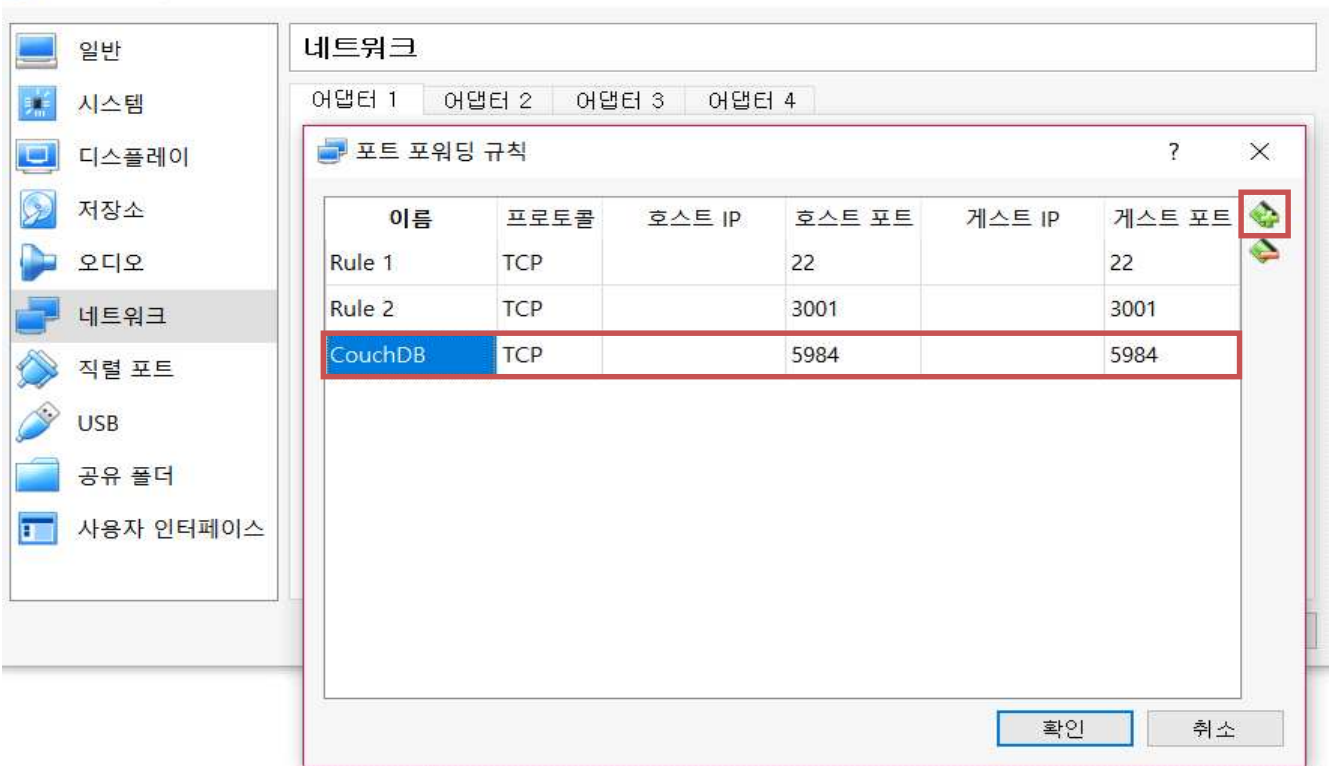
- \$ **peer chaincode install -p chaincodedev/chaincode/sacc2 -n mycc -v 2**로 설치
- \$ **peer chaincode upgrade -n mycc -v 2 -C myc -c '{"Args":["eo","10"]}'**로 업그레이드
: instantiate해도 되긴하는데 기존에 각 node별로 전파된 것을 다 수정해줘야하므로 upgrade 추천
(굳이 새로 instantiate 하지 않아도 됨)
- \$ **peer chaincode list —installed / peer chaincode list —instantiated -C myc**
: 기존에 설치된 모든 내용과 현재 instance화 된 최근 정보가 뜸
- 이후에는 **invoke(set, get, transfer, delete)**를 이용해 잘 작동하는지 실습!!

2. Couch DB로 실습하기

- 최초 ubuntu server의 설정 변경(port번호 추가)

: 맨 마지막에 localhost:5984/_utils로 들어가보면 couchDB로 만든 서버 확인 가능!

ubuntu01_up_1 - 설정



- \$ **docker ps -a**로 아무것도 없는 것 확인
- \$ **rmate -p 52698 marbles_chaincode.go**를 통해 해당 파일 띄우기
- CRUD(insert / select / update / delete)
put state get state del state
- 실습 전에 코드 분석 먼저(marbles_chaincode.go)

```
// === Save marble to state ===  
err = stub.PutState(marbleName, marbleJSONasBytes)  
if err != nil {  
    return shim.Error(err.Error())  
}
```

: 뒤에 넣는 값은 JSON형식의 Byte 타입을 그대로 넣겠다

```
marbleJSONasBytes, err := json.Marshal(marble)
```

: Marshal이란 함수는 JSON형식의 format을 Byte 배열로 바꿔주는 함수(↔UnMarshal)

: 이렇게하면 CouchDB에 그대로 document가 들어갈 수 있음!

이는 굉장히 많은 의미를 가짐

(데이터를 검색할 수 있는 많은 방법을 제공하고, data를 중첩적으로 관리할 수 있음)

```
err := stub.PutState(args[0], []byte(args[1]))  
if err != nil {  
    return shim.Error(fmt.Sprintf("Failed to create asset: %s", args[0]))  
}
```

: 기존 예제(sacc2)에서는 이렇게 PutState 뒤에 []byte(args[1]) 값을 넣었음!! -> levelDB 사용한 것

```
err = json.Unmarshal([]byte(valAsBytes), &marbleJSON)
```

: GetState로 받아온 값을 Unmarshal을 통해 byte배열로 바꿔줌

- fabric-samples/first-network 디렉토리 안에는 다음과 같은 파일들이 있고, 네트워크 기동에 이용!

```
guru@guru:~/fabric-samples/first-network$ ll
total 120
drwxrwxr-x 8 guru guru 4096 Nov 16 16:24 ./
drwxrwxr-x 14 guru guru 4096 Nov 15 13:20 ../
drwxrwxr-x 2 guru guru 4096 Nov 14 13:42 base/
-rwxrwxr-x 1 guru guru 20636 Nov 14 13:42 byfn.sh*
drwxrwxr-x 2 guru guru 4096 Nov 16 14:14 channel-artifacts/
-rw-rw-r-- 1 guru guru 12265 Nov 14 13:42 configtx.yaml
drwxr-xr-x 4 guru guru 4096 Nov 16 14:14 crypto-config/
-rw-rw-r-- 1 guru guru 3906 Nov 14 13:42 crypto-config.yaml
-rw-rw-r-- 1 guru guru 2971 Nov 14 13:42 docker-compose-cli.yaml
-rw-rw-r-- 1 guru guru 2345 Nov 14 13:42 docker-compose-couch-org3.yaml
-rw-rw-r-- 1 guru guru 4560 Nov 14 13:42 docker-compose-couch.yaml
-rw-rw-r-- 1 guru guru 2883 Nov 14 13:42 docker-compose-e2e-template.yaml
-rw-rw-r-- 1 guru guru 3801 Nov 14 13:42 docker-compose-org3.yaml
-rw-rw-r-- 1 guru guru 42 Nov 14 13:42 .env
-rwxrwxr-x 1 guru guru 10409 Nov 14 13:42 eyfn.sh*
drwxrwxr-x 14 guru guru 4096 Nov 16 16:25 fabric-samples/
-rw-rw-r-- 1 guru guru 118 Nov 14 13:42 .gitignore
drwxrwxr-x 2 guru guru 4096 Nov 14 13:42 org3-artifacts/
-rw-rw-r-- 1 guru guru 335 Nov 14 13:42 README.md
drwxrwxr-x 2 guru guru 4096 Nov 14 13:42 scripts/
```

- 네트워크를 기동하기 전에 교재 p128부터 나와있는 인증서, genesis블록, 채널생성, 앵커피어 등록 등의 작업을 실시. 완료 이후 네트워크 기동(아래)

- **docker-compose -f docker-compose-cli.yaml -f docker-compose-couch.yaml up -d**로 네트워크 기동
: 네트워크 기동은 cli를 이용해서 하고 DB는 couch를 사용하겠다!(그래서 2개 / default는 levelDB)
: 가장 쉬운 네트워크 기동 : byfn.sh(levelDB)이지만, peer를 추가하고 channel에 peer들을 가입시키기 위해서 docker-compose-couch.yaml을 사용해 네트워크 기동! **교재 p146 참고해서 실습**
: 네트워크 기동하면 최초에는 create ~~ 뜨지만 내용 수정 후 다시 기동하면 up-to-date 뜸!

```
guru@guru:~/fabric-samples/first-network$ docker-compose -f docker-compose-cli.yaml -f docker-compose-couch.yaml up -d
Creating network "net_byfn" with the default driver
Creating volume "net_peer0.org2.example.com" with default driver
Creating volume "net_peer1.org2.example.com" with default driver
Creating volume "net_peer1.org1.example.com" with default driver
Creating volume "net_peer0.org1.example.com" with default driver
Creating volume "net_orderer.example.com" with default driver
Creating couchdb0
Creating couchdb1
Creating orderer.example.com
Creating couchdb2
Creating couchdb3
Creating peer0.org2.example.com
Creating peer1.org2.example.com
Creating peer1.org1.example.com
Creating peer0.org1.example.com
Creating cli
```

- 만약 잘못했을 경우??

\$ **rm -rf crypto-config**로 인증서 파일을 모두 지우고, \$ **cd channel-artifacts/**로 이동해

\$ **rm -rf *** 내용을 모두 지움(디렉토리는 반드시! 남겨둘 것)

또는, first_network에서 \$ **./byfn.sh -m down** 하고 재시작!(CA생성~)해도 됨.

- **\$docker exec -it cli bash**이용 docker로 들어가서 peer에 install 및 instance화 작업 실시.

: 강사님 [github](https://github.com/joneconsulting/bc/blob/master/BYFN_couchdb_cmd.txt#L7) 참고(https://github.com/joneconsulting/bc/blob/master/BYFN_couchdb_cmd.txt#L7)

3. Private Data

- 비트코인, 이더리움에서 사용했던 consensus(합의)라는 단어 대신 policy(정책)이란 단어 사용!
- blockToLive라는 개념을 도입해 일정블록 생성 이후 데이터 삭제 가능하게 만듦(보호를 위해)
- collectionMarbles, collectionMarblesPrivateDetails라는 JSON파일

- fabric-samples/chaincode/marbles02_private 밑에 있는 파일 두 개 열기(.json, .go)

```
$ rmate -p 52698 collections_config.json
```

```
$ rmate -p 52698 marbles_chaincode_private.go
```

```
{ } collections_config.json • marbles_chaincode_private.go
1  [
2  {
3      "name": "collectionMarbles",
4      "policy": "OR('Org1MSP.member', 'Org2MSP.member')",
5      "requiredPeerCount": 0,
6      "maxPeerCount": 3,
7      "blockToLive": 1000000
8  },
9  {
10     "name": "collectionMarblePrivateDetails"
11     "policy": "OR('Org1MSP.member')",
12     "requiredPeerCount": 0,
13     "maxPeerCount": 3,
14     "blockToLive": 3
15 }
16 ]
```

- \$./byfn.sh down으로 모든 채널,서버 죽이고
- \$./byfn.sh up -c mychannel -s couchdb를 이용해 couchdb를 갖는 새로운 시스템 기동
- \$ docker exec -it cli bash를 이용해 docker로 이동!!
interpret terminal
- 각 peer별 체인코드(marblesp) install 후 instantiate(이놈은 한 번만!)
기존과 똑같이 치고 마지막에 --collections-config 옵션 추가(젤 중요★)
- 자주 나오는 포트번호의 의미? 7050(orderer), 그 이상(각종 peer)
- \$ 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","marble1","blue","35","tom","99"]}'

```
func (t *SimpleChaincode) initMarble(stub shim.ChaincodeStubInterface, args []string) pb.Response {
    var err error

    // 0-name 1-color 2-size 3-owner 4-price
    // "asdf", "blue", "35", "bob", "99"
    if len(args) != 5 {
        return shim.Error("Incorrect number of arguments. Expecting 5")
    }
}
```

: initMarble이란 함수는 5개의 인자를 받음
(맨 위에 설정한 struct를 보면 마지막 price는 marblePrivateDetails의 데이터임을 알 수 있음)

```
type marble struct {
    ObjectType string `json:"docType"`
    Name        string `json:"name"`
    Color       string `json:"color"`
    Size        int    `json:"size"`
    Owner       string `json:"owner"`
}

type marblePrivateDetails struct {
    ObjectType string `json:"docType"`
    Name        string `json:"name"`
    Price       int    `json:"price"`
}
```

코드 내에 PutState를 보면 4개 / 1개 값 따로 입력!!

: 어쨌든 위에 코드 실행(invoke)하면 아래 결과 표시

```
2018-11-20 07:09:32.224 UTC [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke successful. result: status:200
```

- initMarble가 잘 실행되었는지 Query문을 통해 확인(Org1에서 2개 → Org2에서 2개)

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

```
root@10cc4270dc10:/opt/gopath/src/github.com/hyperledger/fabric/peer# peer chaincode query -C mychannel -n marblesp -c '{"Args":["readMarble","marble1"]}'
{"color":"blue","docType":"marble","name":"marble1","owner":"tom","size":35}
```

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

```
root@10cc4270dc10:/opt/gopath/src/github.com/hyperledger/fabric/peer# peer chaincode query -C mychannel -n marblesp -c '{"Args":["readMarblePrivateDetails","marble1"]}'
{"docType":"marblePrivateDetails","name":"marble1","price":99}
```

: Org1에서는 둘 다 잘 나오지만, Org2는 PrivateDetails(즉, price 정보)를 볼 수 없기 때문에 에러 발생!

```
root@10cc4270dc10:/opt/gopath/src/github.com/hyperledger/fabric/peer# 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: dbcb417c2e11014fddff9c338de71fdab9fbac7a9e01bfff8e481ae529150527c: private data matching public hash version is not available. Public hash version = &version.Height{BlockNum:0x6, TxNum:0x0}, Private data version = (*version.Height)(nil)\\"}
```

- 수명을 가지는 Private Data

: 새 터미널 창을 띄우고 \$ docker logs peer0.org1.example.com 2>&1 | grep -i -a -E 'private|pvt|privdata' 입력해 현재 몇 개의 블록이 만들어졌는지 확인

```
guru@guru:~$ docker logs peer0.org1.example.com 2>&1 | grep -i -a -E 'private|pvt|privdata'
2018-11-20 06:02:00.749 UTC [kvledger] CommitWithPvtData -> INFO 02e [mychannel] Committed block [0] with 1 transaction(s) in 110ms (state_validation=0ms block_commit=15ms state_commit=92ms)
2018-11-20 06:02:13.871 UTC [gossip/privdata] StoreBlock -> INFO 03b [mychannel] Received block [1] from buffer
2018-11-20 06:02:13.935 UTC [kvledger] CommitWithPvtData -> INFO 042 [mychannel] Committed block [1] with 1 transaction(s) in 38ms (state_validation=0ms block_commit=16ms state_commit=19ms)
2018-11-20 06:02:17.058 UTC [gossip/privdata] StoreBlock -> INFO 043 [mychannel] Received block [2] from buffer
2018-11-20 06:02:17.222 UTC [kvledger] CommitWithPvtData -> INFO 04a [mychannel] Committed block [2] with 1 transaction(s) in 124ms (state_validation=0ms block_commit=58ms state_commit=63ms)
2018-11-20 06:02:57.835 UTC [gossip/privdata] StoreBlock -> INFO 04f [mychannel] Received block [3] from buffer
2018-11-20 06:02:58.178 UTC [kvledger] CommitWithPvtData -> INFO 053 [mychannel] Committed block [3] with 1 transaction(s) in 286ms (state_validation=17ms block_commit=62ms state_commit=202ms)
2018-11-20 06:03:39.669 UTC [gossip/privdata] StoreBlock -> INFO 05a [mychannel] Received block [4] from buffer
2018-11-20 06:03:39.818 UTC [kvledger] CommitWithPvtData -> INFO 05c [mychannel] Committed block [4] with 1 transaction(s) in 139ms (state_validation=50ms block_commit=39ms state_commit=46ms)
2018-11-20 06:04:02.137 UTC [gossip/privdata] StoreBlock -> INFO 060 [mychannel] Received block [5] from buffer
2018-11-20 06:04:02.829 UTC [kvledger] CommitWithPvtData -> INFO 065 [mychannel] Committed block [5] with 1 transaction(s) in 655ms (state_validation=499ms block_commit=53ms state_commit=93ms)
2018-11-20 07:09:34.254 UTC [gossip/privdata] StoreBlock -> INFO 06d [mychannel] Received block [6] from buffer
2018-11-20 07:09:34.496 UTC [couchdb] CreateDatabaseIfNotExist -> INFO 06f Created state database mychannel_marblesp$collection$marble$private$details
2018-11-20 07:09:34.572 UTC [couchdb] CreateDatabaseIfNotExist -> INFO 070 Created state database mychannel_marblesp$collection$marble$private$details
2018-11-20 07:09:34.877 UTC [kvledger] CommitWithPvtData -> INFO 071 [mychannel] Committed block [6] with 1 transaction(s) in 560ms (state_validation=42ms block_commit=70ms state_commit=440ms)
```

: 현재까지 6개의 블록이 생성됨! 'collections_config.json' 파일에 명시한 내용에 의해 앞으로 3개의 블록을 더 생성하면 price라는 data는 사라질 것!(실습을 위해 invoke - initMarble 3번 실시)

: initMarble(7번 블록) → transferMarble(8번 블록) → transferMarble(9번 블록)

할 때 까지 query문 실행해보면 이상 없이 'price:99'라는 결과가 호출!!

: transferMarble(10번 블록) 마지막 실행 후 query 날려보면 데이터 삭제된 것을 알 수 있음.

```
root@10cc4270dc10:/opt/gopath/src/github.com/hyperledger/fabric/peer# peer chaincode query -C mychannel -n marblesp -c '{"Args":["readMarblePrivateDetails","marble1"]}'
Error: endorsement failure during query. response: status:500 message:{"Error":":Marble private details does not exist: marble1\\"}
2018-11-20 08:18:15.825 UTC [gossip/privdata] StoreBlock -> INFO 08f [mychannel] Received block [10] from buffer
2018-11-20 08:18:16.378 UTC [kvledger] CommitWithPvtData -> INFO 091 [mychannel] Committed block [10] with 1 transaction(s) in 513ms (state_validation=38ms block_commit=18ms state_commit=237ms)
```

: channel state가 아닌 private state에 저장되어 있다가 사라짐(json파일에 따로 policy를 설정해주면 사용 가능!) sideDB(peer가 가지고 있는 개별적인 공간)