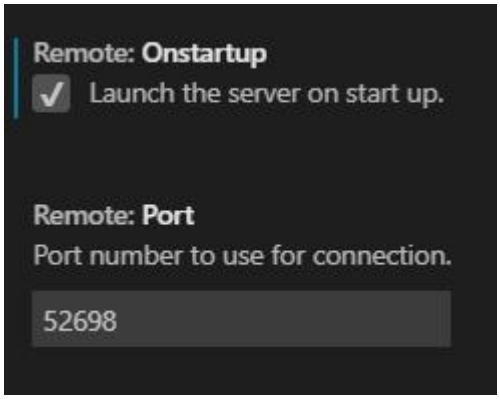
< 하이퍼레져 페브릭 환경설정 >

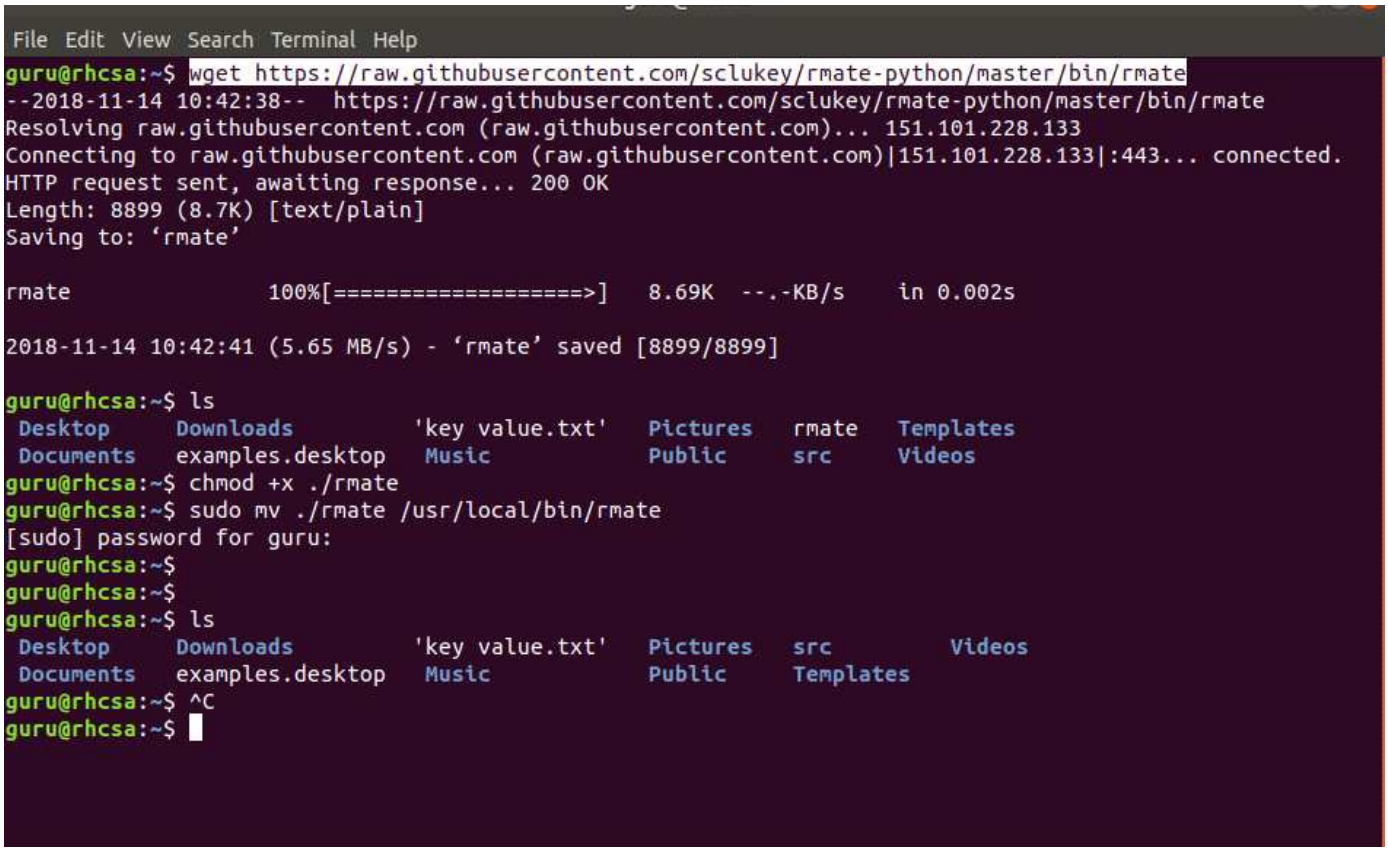
2018. 11. 14(수)

1. 하이퍼레져 페브릭 실습준비

- visualcode에서 'rmate' 깔고 → setting에서 'remote' 검색 후 활성화 → port번호 기억(52698)



- VM켜서 rmate 깔고 mode바꿔서 폴더 옮기기!!(ubuntu01 사용)



- visualcode에서 **ctrl+`(~)** 누르면 터미널 창 뜸!!

<VScode 에서>

* ssh -R 52698:localhost:52698 user_id@127.0.0.1
  ssh(ubuntu에 접속하기 위한 명령어)로 52698이라는 port에 들어가라 / 뒤에는 내 서버 리눅스랑 내꺼랑 두 개 연결 해준 것!!

<Ubuntu 에서>

* rmate –p 52698 파일명
  VS code로 파일 전송하는 명령어

- **만약, ubuntu로 접속하는게 오류난다면??** https://cpuu.postype.com/post/30065 참고

```
어다희@DESKTOP-1CDR712 MINGW64 /c/work_blockchain
$ ssh -R 52698:localhost:52698 guru@127.0.0.1
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the ECDSA key sent by the remote host is
SHA256:9o2yXXREtAlm9z16oFSL2E4XYPMTVpd0P/O2gZjw2Zo.
Please contact your system administrator.
Add correct host key in /c/Users/\354\226\264\353\213\244\355\235\254/.ssh/known_hosts to get rid of this message.
Offending ECDSA key in /c/Users/\354\226\264\353\213\244\355\235\254/.ssh/known_hosts:1
ECDSA host key for 127.0.0.1 has changed and you have requested strict checking.
Host key verification failed.
```

(SSH 접속시 RSA 공유키 충돌문제 발생 → keygen으로 초기화 해주고 다시 해 볼 것)
기존에 127.0.0.1이라는 ip로 접속한 적이 있는 서버와 RSA공유키를 교환한 상태에서 서버가 바뀌었기 때문에 발생!
위 경고 메시지는 Man in the Middle Attack 이라는 일명 '중간자 공격'에 대해 경고한다. 즉, 기존에 서버가 알고있던 정보를 찾아서 따라갔더니- 기존과는 전혀 다른 서버로 접속되었다는 것이다.
이를 해결하기 위해 **ssh-keygen –R 127.0.0.1** 이라는 명령어를 통해 초기화시켜준다!

- [하이퍼레져 페브릭을 사용하기 위해 깔아야 할 파일들]

1. curl
$ curl -fsSL https://get.docker.com/ | sudo sh
$ sudo usermod –aG docker guru

2. go lang
$ sudo apt-get install –y golang → version확인 후 1.10보다 낮으면 업데이트
$ wget https://dl.google.com/go/go1.10.linux-amd64.tar.gz
$ sudo tar –xvf go1.10.linux-amd64.tar.gz → $ sudo mv go /usr/local →
$ export GOROOT=/usr/local/go → which go (다른 경로 go를 지우고 재부팅)
$export PATH=$PATH:$GOROOT/bin: → go version(1.10 확인)

3. nodejs
$ curl -sL https://deb.nodesource.com/setup_8.x | sudo -E bash -
$ sudo apt-get install –y nodejs → node —version(8.12 확인)

4. python
$ sudo apt-get install –y python

5. docker
$ sudo apt-get install –y docker

6. docker-compose
update와 upgrade 해주고 → $ sudo apt-get install –y docker-compose
--version으로 각각의 버전 확인(도커는 17, 컴포즈는 14) → docker images해서 안 뜨면
logout해서 다시 로그인!! ssh ~~ → docker images해서 빈 파일 확인!!

7. hyperledge fabric($ curl –sSL http://bit.ly/2ysbOFE | bash –s 1.3.0)
사실상 페브릭을 까는 코드는 위에 7번 한줄!!(나머지는 docker를 사용하기 위한 환경설정 수준)
완료되었으면 $ docker images 해서 잘 설치되었는지 확인!!

curl은 우리가 앞서 사용한 postman과 같이 docker에서 연결을 도와주는 역할을 하는 놈

```
guru@guru:~$ docker images
REPOSITORY                    TAG                 IMAGE ID            CREATED             SIZE
mysql                         5.7                 702fb0b7837f        2 weeks ago         372MB
hyperledger/fabric-javaenv    1.3.0               2476cefaf833        4 weeks ago         1.7GB
hyperledger/fabric-javaenv    latest              2476cefaf833        4 weeks ago         1.7GB
hyperledger/fabric-ccenv      1.3.0               953124d80237        4 weeks ago         1.38GB
hyperledger/fabric-ccenv      latest              953124d80237        4 weeks ago         1.38GB
hyperledger/fabric-orderer    1.3.0               f430f581b46b        4 weeks ago         145MB
hyperledger/fabric-orderer    latest              f430f581b46b        4 weeks ago         145MB
hyperledger/fabric-peer       1.3.0               f3ea63abddaa        4 weeks ago         151MB
hyperledger/fabric-peer       latest              f3ea63abddaa        4 weeks ago         151MB
```

- [docker image 실습]

$ docker run –d –p 3306:3306 –e MYSQL_ALLOW_EMPTY_PASSWORD=true —name mysql mysql:5.7

: 도커를 실행하는데 데몬버전(백에서) 실행하고 포트는 3306(HOST PC와 DOCKER를 연결해주는)을 사용, 환경변수, 비밀번호는 없이 사용하고 이름은 mysql이라 부여하겠다. 버전은 mysql:5.7 사용!

$ docker exec –it mysql bash

```
guru@guru:~$ docker exec -it mysql bash
root@ce1e467f18e0:/#
```

(코드암기)

: 도커로 접속(exec)실행하는데 –it(console모드로)에서 mysql이란 이름의 bash셸 모드로 실행 (docker 이미지가 설치된 아이디로 들어가짐!)

$ mysql –h127.0.0.1 –uroot

: 로 로그인하겠다(위에 비번 설정 안했으니까 그냥 들어가짐) 이후 SELECT문 등 사용 가능!!

```
root@ce1e467f18e0:/# mysql -h127.0.0.1 -uroot
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.7.24 MySQL Community Server (GPL)

Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

- [docker 명령어]

1. $ docker ps : process status 현재 구동중인 프로세스 띄워줌
2. $ docker stop [ID] : 실행중인 프로세스 멈추기
   ex) $ docker stop $(docker ps -qa) && docker rm $(docker ps -qa)
3. $ docker rm [ID] : 프로세스 삭제
4. $ docker image : 도커의 이미지들 싹 띄워라
5. $ docker rmi [ID] : 관련 이미지 삭제

2. 블록체인 적용

- 공통데이터이며, Transaction이 많이 발생하지 않는다면 블록체인 도입해 볼만 함!
- Fabric은 암호화폐를 요구하지 않음(=보상이 없음)
  그 안에 참여하는 기업에 한해서는 보상 없이도 운영하겠다*(허가형 네트워크)*
  합의는 SOLO, Kafka, SBFT 사용
  Fabric에서의 스마트컨트렉트는 Chaincode(이더리움의 Solidity로 구현한 것과 같음)
  하이퍼레져 != 하이퍼레져페브릭(하이퍼레져에 블록체인 기술을 도입 한 것이 페브릭!)
- 하이퍼레져 페브릭은 IBM사에서 개발! 약 2~3만줄의 코드를 Linux에 기부
  리눅스에서 '하이퍼레져 프로젝트' 진행! Fabric은 많은 프레임워크 중의 하나!
  IBM으로부터 Hyperledge에 기부된 첫 번째 프로젝트로 현재는 LINUX에서 관리
- world state라는 개념이 등장(HF는 bc나 eth와 다르게 모든 데이터를 블록에 저장하지 않음. 현재 데이터의 상태를 저장할 수 있도록 하는 상태DB를 제공 => RDBMS가 아닌 DB)
  LevelDB와 CouchDB
  map형식의 key:value가 들어감 noSQL에서 나옴
- chaincode = smart-contract = 최근 상태를 가져오는 것
- 블록의 구조는 비트코인과 거의 유사하지만, KVS(Key Value Store)의 hash값도 저장되는 것이 특징!(world state라는 DB의 hash값)
- 체인코드(CC)는 모든 검증노드에 배포됨 → 검증노드는 CC를 docker컨테이너에 배포하고 실행
- 하이퍼레져 패브릭의 트랜잭션 = 체인코드(스마트컨트렉트)의 실행
- 자산의 최신 상태는 Key-Value Store에 저장함 => couchDB등을 사용
- 계약 조건이나 규칙을 비즈니스 로직으로 스마트계약에 구현함으로써 항상 합의 된 규칙에 따라 처리됨
- 참여자(검증노드) : 분산원장, CC, KVS를 가짐 (→Endorser라 부름)
- IBM에서 하이퍼레져 페브릭의 무결성을 검증하기 위해서는 최소 4개의 노드가 필요하며, 15개의 노드가 가장 적합하다고 공지함.(그 이상도 가능은 한데 15개가 적절하단 뜻)
- 데이터를 공유해야하는데 공유하기 싫은 집단들이 블록체인 기술을 사용(금융, IT, 의료 등)

```
guru@guru:~/fabric-samples/fabcar$ ll
total 40
drwxrwxr-x  2 guru guru 4096 Nov 14 13:55 ./
drwxrwxr-x 14 guru guru 4096 Nov 14 14:13 ../
-rw-rw-r--  1 guru guru 2809 Nov 14 13:55 enrollAdmin.js
-rw-rw-r--  1 guru guru   50 Nov 14 13:55 .gitignore
-rw-rw-r--  1 guru guru 6333 Nov 14 13:55 invoke.js
-rw-rw-r--  1 guru guru  533 Nov 14 13:55 package.json
-rw-rw-r--  1 guru guru 2606 Nov 14 13:55 query.js
-rw-rw-r--  1 guru guru 3147 Nov 14 13:55 registerUser.js
-rwxrwxr-x  1 guru guru 2070 Nov 14 13:55 startFabric.sh*
```

- 'fabcar' 디렉토리는 전부 nodejs로 만들었기에 구성이 .js로 되어있음ㅎㅎ → $ cd fabcar 가서

```
guru@guru:~/fabric-samples/fabcar$ rmate -p 52698 package.json
```

로 package.json 띄우기!!

```
test.py  Blockchain.py  openapi.py  voting.sol  package.json
{
    "name": "fabcar",
    "version": "1.0.0",
    "description": "Hyperledger Fabric Car Sample Application",
    "main": "fabcar.js",
    "scripts": {
        "test": "echo \"Error: no test specified\" && exit 1"
    },
    "dependencies": {
        "fabric-ca-client": "~1.3.0",
        "fabric-client": "~1.3.0",
        "grpc": "^1.6.0"
    },
    "author": "Anthony O'Dowd",
    "license": "Apache-2.0",
    "keywords": [
        "Hyperledger",
        "Fabric",
        "Car",
        "Sample",
        "Application"
```

dependencies 안에 있는 놈들이 젤 중요!!

이 파일을 실행하기 위해 필요한 파일들!
(모르는 것들을 각자 설치 할 수 없음)
$ npm install -g란 명령어를 써주면 알아서 필요파일들 설치(파이썬의 pip과 유사)

```
guru@guru:~/fabric-samples/fabcar$ ./startFabric.sh
```
로 띄우면 6개가 나와야함(docker images)

if) 안될 경우? $ docker stop $(docker ps –qa) && docker rm $(docker ps –qa) 하고 다시 start!!
컨테이너 간 꼬여서 error 발생한 것이기 때문에 지우고 다시 install npm 해주기

- **$ node enrollAdmin.js** 로 관리자 생성

```
docker-compose -f docker-compose.yml up -d ca.example.com orderer.example.com peer0.org1.example.com couchdb
Creating network "net_basic" with the default driver
Creating orderer.example.com
Creating couchdb
Creating ca.example.com
Creating peer0.org1.example.com

# wait for Hyperledger Fabric to start
# incase of errors when running later commands, issue export FABRIC_START_TIMEOUT=<larger number>
export FABRIC_START_TIMEOUT=10
#echo ${FABRIC_START_TIMEOUT}
sleep ${FABRIC_START_TIMEOUT}

# Create the channel
docker exec -e "CORE_PEER_LOCALMSPID=Org1MSP" -e "CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/msp/users/Admin@org1.example.com/msp" peer0.org1.example.com peer channel create -o orderer.examp
le.com:7050 -c mychannel -f /etc/hyperledger/configtx/channel.tx
2018-11-14 07:58:29.501 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2018-11-14 07:58:29.544 UTC [cli/common] readBlock -> INFO 002 Received block: 0
# Join peer0.org1.example.com to the channel.
docker exec -e "CORE_PEER_LOCALMSPID=Org1MSP" -e "CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/msp/users/Admin@org1.example.com/msp" peer0.org1.example.com peer channel join -b mychannel.block

2018-11-14 07:58:29.921 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2018-11-14 07:58:30.059 UTC [channelCmd] executeJoin -> INFO 002 Successfully submitted proposal to join channel
Creating cli
2018-11-14 07:58:31.468 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 001 Using default escc
2018-11-14 07:58:31.468 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 002 Using default vscc
2018-11-14 07:58:31.674 UTC [chaincodeCmd] install -> INFO 003 Installed remotely response:<status:200 payload:"OK" >
2018-11-14 07:58:32.059 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 001 Using default escc
2018-11-14 07:58:32.060 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 002 Using default vscc
2018-11-14 07:58:42.914 UTC [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 003 Chaincode invoke successful. result: status:200
```

이렇게 해서 페브릭이 시작되고 $docker ps해보면 6개의 컨테이너가 활성화됨!!
시작되는 과정을 살펴보면, network는 'net_basic'이란 이름으로 만들고, orderer.example.com을 만들었다는 것을 알 수 있음.
시간이 조금 오래 걸리는 이유는 chaincode를 install하고 invoke하기 때문!

- **$ node enrollAdmin.js** 해보면 Successfully enrolled admin user "admin" 뜨면서 성공됨!!
관리자가 생성된 것!
- 유저 만들기(successfully 뜸)

```
guru@guru:~/fabric-samples/fabcar$ node registerUser.js
 Store path:/home/guru/fabric-samples/fabcar/hfc-key-store
(node:5916) DeprecationWarning: grpc.load: Use the @grpc/proto-loader module with grpc.loadPackageDefinition instead
Successfully loaded admin from persistence
Successfully registered user1 - secret:weenSSipJfCB
Successfully enrolled member user "user1"
User1 was successfully registered and enrolled and is ready to interact with the fabric network
```

- 내부 데이터 확인(10개가 있음)

```
guru@guru:~/fabric-samples/fabcar$ node query.js
Store path:/home/guru/fabric-samples/fabcar/hfc-key-store
(node:5931) DeprecationWarning: grpc.load: Use the @grpc/proto-loader module with grpc.loadPackageDefinition instead
Successfully loaded user1 from persistence
Query has completed, checking results
Response is  [{"Key":"CAR0", "Record":{"colour":"blue","make":"Toyota","model":"Prius","owner":"Tomoko"}},{"Key":"CAR1", "Record":{"colour":"red","make":"Ford","model":"Mustang","owner":"B
rad"}},{"Key":"CAR2", "Record":{"colour":"green","make":"Hyundai","model":"Tucson","owner":"Jin Soo"}},{"Key":"CAR3", "Record":{"colour":"yellow","make":"Volkswagen","model":"Passat","owne
r":"Max"}},{"Key":"CAR4", "Record":{"colour":"black","make":"Tesla","model":"S","owner":"Adriana"}},{"Key":"CAR5", "Record":{"colour":"purple","make":"Peugeot","model":"205","owner":"Miche
l"}},{"Key":"CAR6", "Record":{"colour":"white","make":"Chery","model":"S22L","owner":"Aarav"}},{"Key":"CAR7", "Record":{"colour":"violet","make":"Fiat","model":"Punto","owner":"Pari"}},{"K
ey":"CAR8", "Record":{"colour":"indigo","make":"Tata","model":"Nano","owner":"Valeria"}},{"Key":"CAR9", "Record":{"colour":"brown","make":"Holden","model":"Barina","owner":"Shotaro"}}]
```

$ rmate –p 52698 query.js로 띄워서 확인 후 request의 함수내용 수정(fcn, args) => client 단

```
52      // queryCar chaincode function - requires 1 argument, ex: args: ['CAR4'],
53      // queryAllCars chaincode function - requires no arguments , ex: args: [''],
54      const request = {
55          //targets : --- letting this default to the peers assigned to the channel
56          chaincodeId: 'fabcar',
57          fcn: 'queryCar',
58          args: ['CAR4']
59      };
```

- 다시 **$ node query.js** 실행하면 이렇게 하나만 응답나옴(args를 'CAR4' 하나만 지정했기 때문)

```
guru@guru:~/fabric-samples/fabcar$ node query.js
Store path:/home/guru/fabric-samples/fabcar/hfc-key-store
(node:5987) DeprecationWarning: grpc.load: Use the @grpc/proto-loader module with grpc.loadPackageDefinition instead
Successfully loaded user1 from persistence
Query has completed, checking results
Response is  {"colour":"black","make":"Tesla","model":"S","owner":"Adriana"}
```

- fabcar는 node.js(JS)와 java와 golang을 지원함(client) -> Django로 만들어서 화면 구현
  fabcar.js파일을 열어보면 안에 몇 가지 함수가 있음(queryAllCars 등등)

- **$ rmate –p 52698 invoke.js** 띄워서 64, 65번 줄 추가 및 수정!!

```
61      var request = {
62          //targets: let default to the peer assigned to the client
63          chaincodeId: 'fabcar',
64          fcn: 'createCar',
65          args: ['CAR12', 'Honda', 'Accord', 'Black', 'DAHEE'],
66          chainId: 'mychannel',
67          txId: tx_id
68      };
```

- **$ node invoke.js** 하면 데이터가 성공적으로 invoke 됨
- **$ node query.js** 해서 정상적으로 블록에 반영되었는지 확인(이전에 query.js파일에 args를 CAR12로 바꿔주기)

```
guru@guru:~/fabric-samples/fabcar$ node query.js
Store path:/home/guru/fabric-samples/fabcar/hfc-key-store
(node:6017) DeprecationWarning: grpc.load: Use the @grpc/proto-loader module with grpc.loadPackageDefinition instead
Successfully loaded user1 from persistence
Query has completed, checking results
Response is  {"colour":"Black","make":"Honda","model":"Accord","owner":"DAHEE"}
```

- 한 번 더 실습을 위해 invoke.js 파일내용 수정

```
61      var request = {
62          //targets: let default to the peer assigned to the client
63          chaincodeId: 'fabcar',
64          fcn: 'changeCarOwner',
65          args: ['CAR12', 'Barry'],
66          chainId: 'mychannel',
67          txId: tx_id
68      };
```

- owner가 변경된 것을 확인!!

```
guru@guru:~/fabric-samples/fabcar$ node query.js
Store path:/home/guru/fabric-samples/fabcar/hfc-key-store
(node:6043) DeprecationWarning: grpc.load: Use the @grpc/proto-loader module with grpc.loadPackageDefinition instead
Successfully loaded user1 from persistence
Query has completed, checking results
Response is  {"colour":"Black","make":"Honda","model":"Accord","owner":"Barry"}
```

- 이렇게 fabric이 제공해주는 함수는 5가지가 있음
  확인을 위해 fabric-samples/chaincode/fabcar/node로 이동(cd명령어)
  ll로 확인해보면 나오는 fabcar.js파일을 rmate로 켜보면??!?

```
async queryCar(stub, args) {…
}

async initLedger(stub, args) {…
}

async createCar(stub, args) {…
}

async queryAllCars(stub, args) {…
}

async changeCarOwner(stub, args) {…
}
;
```

push 명령어로 이루어진 5개의 CC가 있는 것을 확인 가능!!
이 함수 중에서 실습 가능(위에 3가지 해봄)

< 하이퍼레져 페브릭 실습 >

2018. 11. 15(목)

1. 하이퍼레져 페브릭 실습준비

- VM의 우분투 들어가서 로그인(guru / work)해서 서버 기동
  → VisualCode에서 $ ssh 52698:localhost:52698 guru@127.0.0.1로 로그인

```
19  # set PATH so it includes user's private bin directories
20  export GOROOT=/usr/local/go
21  export GOPATH=$HOME/gowork
22  PATH="$HOME/bin:$HOME/.local/bin:$PATH:$GOROOT/bin:"
guru@guru:~$ go get github.com/hyperledger/fabric
```

- 하고 fabric-ca해서 한 번 더! 설치!

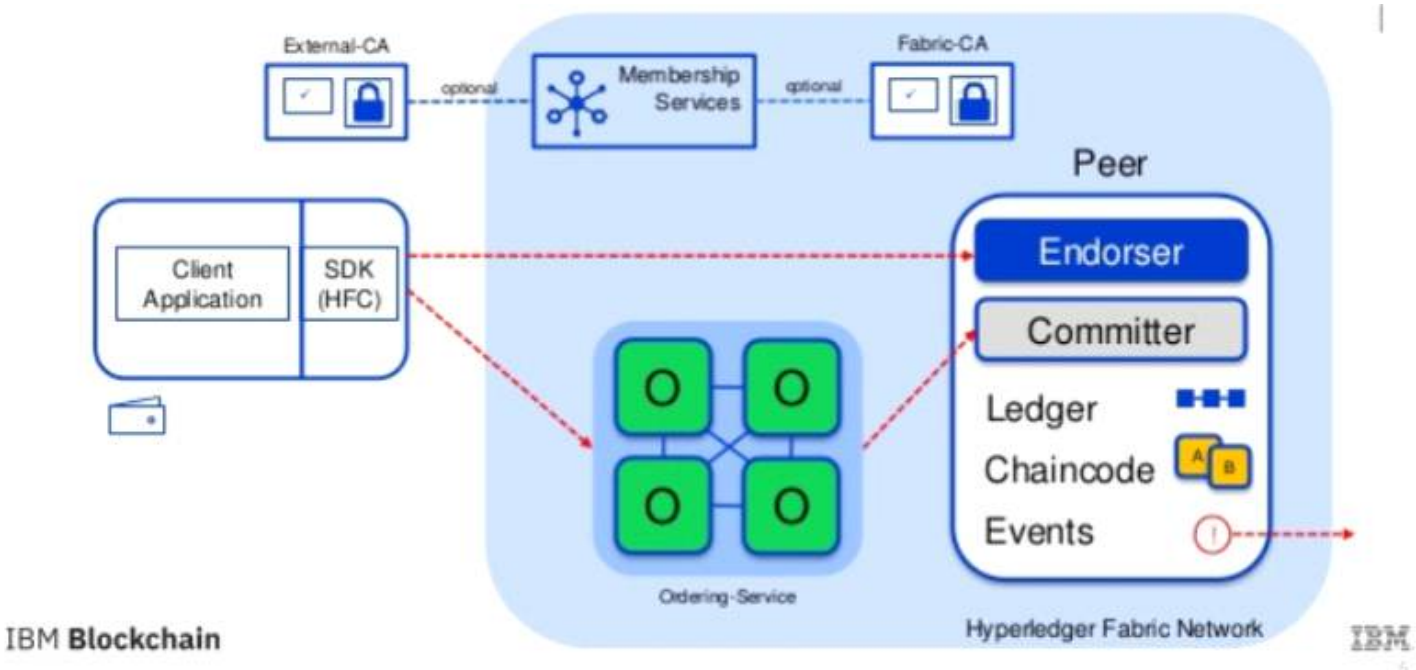
- $ cd $GOPATH → ls해서 src가 있는 것을 확인 →
  $ cd src/github.com/hyperledger/fabric 위치이동 → $ git branch로 'release-1.3' 확인 →
  $ make native docker 실행

- $ cd src/github.com/hyperledger/fabric-ca 위치이동 → $ make docker 실행

2. 블록체인 직업

- 블록체인 개발자 : 어플리케이션, 스마트컨트렉트의 개발.(피어 등에는 관심 ㄴ)
- 블록체인 운영자 : 동작구조에 대해 상세히 파악해야 함.(peer, consensus, security)
  (반드시 블록체인 개발을 할 수 있어야 하는 것은 아님 / 코딩 ㄴ)
- 블록체인 아키텍트 : 비즈니스 디자인 등 큰 설계를 해야 함.

3. Hyperledger Fabric V1 Architecture



: Fabric-CA(인증서 발급 기관 : Certificate Authority)
$ cd fabric-samples/fabcar/hfc-key-store해서 들어가 보면 private과 public key가 생성됨
(외부 인증기관이던, 페브릭 인증기관이던 승인을 받아 시스템에 들어오게되면? KEY가 생성됨!)

: Endorser는 승인을 해주는 역할
SDK가 proposal(어떤 이벤트) 보낸 것에 대해 ChainCode를 실행해보고, 인증서를 검증해서 맞는지 확인(World State 이용)

: Endorser가 검증 후 SDK에 다시 보내주는 이유?
합의를 위해서(엔도서만 아는 것이 아니라 사용하는 참가자 모두에게 내용을 알려주기 위함)

: Ordering-Service에는 여러 개의 채널이 들어올 수 있음
SDK로부터 받은 내용들을 채널별, 시간별 정리 후 Leader에게 보내고 그게 Committer에게 전달 됨!

: Committer가 최종 받은 데이터를 Endorser에게 받은 결과와 같은지 비교를 통해 블록에 기록!

: Committer가 기록하는 것은 원장(Ledger)인데 이는 world state + blockchain으로 이루어짐
'get', 'put', 'delete' 'recorded'

: World State는 levelDB와 couchDB로 구성되어 있음
key:value값만 쓰여있는 DB NoSQL의 종류 중 하나로 테이블이란 개념이 없는 DB(실제 최종 값이 저장됨)
모든 데이터가 아니라 마지막 값(모든 데이터는 어짜피 블록에 쌓임)

4. Fabric Ledger

- blockchain + world state로 구성
- 채널 : peer(committer)와 node(orderer)를 연결해주는 가상공간
  채널을 나누는 등의 작업은 최초 설계 시에 해야 함.(채널을 나누는 이유? 각 체인 별 privacy)

5. Ordering Service

- 트랜잭션들을 블록으로 패키징하여 피어에게 전달하는 역할을 담당.
- 채널을 통해 Ordering Service와 통신
  서로 다른 원장 간 프라이버시를 제공함(채널 별로 또 다른 수첩이 있다고 생각하면 됨), peer는 멀티채널에 참여 할 수 있음
  orderor에 들어올 때는 채널 구분 없이 순서대로 쌓이고 → 최종 원장에 쓰여 질 때 다시 채널별로 원장이 생김
- 설정옵션(SOLO / Kafka)
  개발을 위한 하나의 노드 최소 3개의 노드가 필요(홀수노드 권장)

6. Fabric Peer

- 한 개 이상의 채널에 연결됨(각 채널에 대해서 한 개 이상의 원장을 관리)
- 체인코드는 도커 컨테이너로 분리되어 인스턴스화 되며, 채널을 통해 공유됨

7. Client Application

- 모든 클라이언트는 패브릭 SDK를 사용
- 개인키는 login할 때 쓰이는 정보고, channel은 이 생태계에 들어온 이후 쓰이는 방 번호 같은 개념

8. Transaction Flow(유통업계)

- 어부가 참치를 잡아서 app(웹사이트, 스마트폰)에 로그를 기록하고 디지털 서명을 함 → Endorser peer에게 데이터가 넘어감 → 서명에 대한 검증과 Chain code에 의한 데이터 검증 → 다시 app에 Endorser Transactions 보냄 → app은 O-service에 트랜잭션 보냄 → Orderer는 시간 순으로 순차적 정리를 통해 leader에게 보냄 → leader는 Committer에게 전송 → Committer가 최종 확인 후 채널별 블록에 기록(원장 기입)

9. 실습!!!!!!

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

: 인증서를 새로 만드는데, crypto-config.yaml로 config(구성)해서 만들어라

: 이 명령어를 실행 후 $ ll을 해보면, crypto-config라는 디렉토리가 생성됨!!(내부에는 orderer, peer 생성)

: 생성된 파일 내부로 더 들어가보면 ca/, msp/, peers/, tlsca/ 등 필요 정보들이 전부 생성된 것을 확인 가능!

```
guru@guru:~/fabric-samples/first-network/crypto-config$ ll
total 16
drwxr-xr-x 4 guru guru 4096 Nov 15 18:03 ./
drwxrwxr-x 7 guru guru 4096 Nov 15 18:03 ../
drwxr-xr-x 3 guru guru 4096 Nov 15 18:03 ordererOrganizations/
drwxr-xr-x 4 guru guru 4096 Nov 15 18:03 peerOrganizations/
```

$ FABRIC_CFG_PATH=$PWD(print working directory)

: CFG(ConFigGen)의 경로를 현재경로로 지정

$ ../bin/configtxgen -profile TwoOrgsOrdererGenesis -channelID byfn-sys-channel –outputBlock ./channel-artifacts/genesis.block

: Orderer genesis block 생성(-outputBlock : 제네시스블록이 생성된 경로와 이름)

$ export CHANNEL_NAME=mychannel

: export(리눅스에서 사용하는 환경변수 명령어)를 사용해 CHALLEL_NAME을 쓰면 자동 mychannel을 지정하도록

$ ../bin/configtxgen -profile TwoOrgsChannel –outputCreateChannelTx ./channel-artifacts/channel.tx –channelID $CHANNEL_NAME

: Channel Transaction artifacts 생성

```
guru@guru:~/fabric-samples/first-network$ ../bin/cryptogen generate --config=./crypto-config.yaml
org1.example.com
org2.example.com
guru@guru:~/fabric-samples/first-network$
guru@guru:~/fabric-samples/first-network$
guru@guru:~/fabric-samples/first-network$
guru@guru:~/fabric-samples/first-network$ ../bin/configtxgen -profile TwoOrgsOrdererGenesis -channelID byfn-sys-channel -outputBlock ./channel-artifacts/genesis.block
2018-11-15 15:56:22.047 KST [common/tools/configtxgen] main -> INFO 001 Loading configuration
2018-11-15 15:56:22.233 KST [common/tools/configtxgen] doOutputBlock -> INFO 002 Generating genesis block
2018-11-15 15:56:22.237 KST [common/tools/configtxgen] doOutputBlock -> INFO 003 Writing genesis block
guru@guru:~/fabric-samples/first-network$
guru@guru:~/fabric-samples/first-network$
guru@guru:~/fabric-samples/first-network$
guru@guru:~/fabric-samples/first-network$ export CHANNEL_NAME=mychannel
guru@guru:~/fabric-samples/first-network$ ../bin/configtxgen -profile TwoOrgsChannel -outputCreateChannelTx ./channel-artifacts/channel.tx -channelID $CHANNEL_NAME
2018-11-15 16:00:26.900 KST [common/tools/configtxgen] main -> INFO 001 Loading configuration
2018-11-15 16:00:27.103 KST [common/tools/configtxgen] doOutputChannelCreateTx -> INFO 002 Generating new channel configtx
2018-11-15 16:00:27.110 KST [common/tools/configtxgen] doOutputChannelCreateTx -> INFO 003 Writing new channel tx
```

$ cd ./channel-artifacts/ 에 가보면 파일 2개가 생성됨(channel.tx와 genesis.block)

../bin/configtxgen -profile TwoOrgsChannel –outputAnchorPeersUpdate ./channel-artifacts/Org1MSPanchors.tx -channelID $CHANNEL_NAME -asOrg Org1MSP

```
guru@guru:~/fabric-samples/first-network$ ../bin/configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate ./channel-artifacts/Org1MSPanchors.tx -channelID $CHANNEL_NAME -asOrg Org1MSP
2018-11-15 16:10:21.630 KST [common/tools/configtxgen] main -> INFO 001 Loading configuration
2018-11-15 16:10:21.842 KST [common/tools/configtxgen] doOutputAnchorPeersUpdate -> INFO 002 Generating anchor peer update
2018-11-15 16:10:21.845 KST [common/tools/configtxgen] doOutputAnchorPeersUpdate -> INFO 003 Writing anchor peer update
guru@guru:~/fabric-samples/first-network$
guru@guru:~/fabric-samples/first-network$ ../bin/configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate ./channel-artifacts/Org2MSPanchors.tx -channelID $CHANNEL_NAME -asOrg Org2MSP
2018-11-15 16:10:51.112 KST [common/tools/configtxgen] main -> INFO 001 Loading configuration
2018-11-15 16:10:51.315 KST [common/tools/configtxgen] doOutputAnchorPeersUpdate -> INFO 002 Generating anchor peer update
2018-11-15 16:10:51.318 KST [common/tools/configtxgen] doOutputAnchorPeersUpdate -> INFO 003 Writing anchor peer update
```

: 두 가지 조직(Org1, 2)에 각각 anchor를 지정하는 작업(peer 두 개 중 하나를 앵커로 지정)

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

: Network 기동하면 6개(피어4개, orderer, cli가 켜짐)

```
guru@guru:~/fabric-samples/first-network$ docker-compose -f docker-compose-cli.yaml up -d
Starting peer0.org2.example.com
Starting peer1.org1.example.com
Starting peer0.org1.example.com
Starting orderer.example.com
Starting peer1.org2.example.com
Starting cli
```

docker-compose-cli.yaml파일을 실행하는데 up이라는 명령어를 사용해서 –d(데몬 / 백에서 돌리겠다)로 실행

$ docker ps → 해보면 총 6개의 process가 기동중(peer2개 / orderer / cli)

```
guru@guru:~/fabric-samples/first-network$ docker ps
CONTAINER ID        IMAGE                               COMMAND             CREATED             STATUS              PORTS                                            NAMES
e85aa81f3251        hyperledger/fabric-tools:latest     "/bin/bash"         2 hours ago         Up 18 seconds                                                        cli
66c0e0ba7baa        hyperledger/fabric-orderer:latest   "orderer"           2 hours ago         Up 19 seconds       0.0.0.0:7050->7050/tcp                           orderer.example.com
0cca1bebcb0d        hyperledger/fabric-peer:latest      "peer node start"   2 hours ago         Up 21 seconds       0.0.0.0:7051->7051/tcp, 0.0.0.0:7053->7053/tcp   peer0.org1.example.com
ae91525618bf        hyperledger/fabric-peer:latest      "peer node start"   2 hours ago         Up 20 seconds       0.0.0.0:10051->7051/tcp, 0.0.0.0:10053->7053/tcp peer1.org2.example.com
c4732d1c5129        hyperledger/fabric-peer:latest      "peer node start"   2 hours ago         Up 19 seconds       0.0.0.0:9051->7051/tcp, 0.0.0.0:9053->7053/tcp   peer0.org2.example.com
cd1dca3df30e        hyperledger/fabric-peer:latest      "peer node start"   2 hours ago         Up 20 seconds       0.0.0.0:8051->7051/tcp, 0.0.0.0:8053->7053/tcp   peer1.org1.example.com
```

$ docker exec -it cli bash

: Docker 실행(CLI) why? 전부 도커에서 돌아가므로 ubuntu에서 docker로 이동(guru → peer로 바뀐 것 확인)

```
guru@guru:~/fabric-samples/first-network$ docker exec -it cli bash
root@e85aa81f3251:/opt/gopath/src/github.com/hyperledger/fabric/peer#
```

cd crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem 이 경로 밑에 인증서 파일이 존재!!(cert 파일)

$ export CHANNEL_NAME=mychannel

: docker와 ubuntu는 다르기 때문에 docker에서도 한 번 더 환경설정을 해줘야 함!
: 안 해주면 아래 명령어 작성 시 오류 발생!! 꼭!!!!!써주기

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

: 채널 생성해주는 코드 인증서 이름을 설정하고 해당 경로를 지정해줌(--tls는 대표적인 인증서를 나타내는 옵션)

$ ls 해서 mychnnel.block이 나오면 성공!!

안되면? 2개 지우고 다시

* crypto-config는 rm –rf로 디렉토리 자체를 날려버리고
* channel은 cd로 파일 안에 이동해서 내부 파일만 rm –rf *

< 하이퍼레져 페브릭 실습2 >

2018. 11. 16[금]

1. 합의 알고리즘

- execute → order → validate → update state

kafka(Apache 서비스 중의 하나로 메시지 큐 서비스)로 순차적으로 발생한 event(Tx)에 대해서 시간 순으로 순차적 정렬
합의 관련 내용은 이미 execute 단계에서 endorser가 실시하기 때문에 order는 별도의 검증 과정을 행하지 않음

- https://hyperledger-fabric.readthedocs.io/en/release-1.3/command_ref.html
  위 사이트에서 명령어 관련 옵션 정보들 확인 가능!!(모르면 들어가서 읽어보자)

2. 어제 실습에 이어서(channel join부터)

```
## peer0.org1(4줄 +1)
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
peer channel join –b mychannel.block

## peer1.org1
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=peer1.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/peer1.org1.example.com/tls/ca.crt
peer channel join –b mychannel.block

## peer0.org2
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp
CORE_PEER_ADDRESS=peer0.org2.example.com:7051
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 channel join –b mychannel.block

## peer1.org2
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp
CORE_PEER_ADDRESS=peer1.org2.example.com:7051
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 channel join –b mychannel.block
```

위 명령어를 문단별로 총 4번 입력해보면?
아래와 같이 endorser와 orderer의 연결이 초기화 되고, peer가 성공적으로 채널에 들어가짐!

```
root@f831468920b3:/opt/gopath/src/github.com/hyperledger/fabric/peer# CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
root@f831468920b3:/opt/gopath/src/github.com/hyperledger/fabric/peer# CORE_PEER_ADDRESS=peer1.org1.example.com:7051
root@f831468920b3:/opt/gopath/src/github.com/hyperledger/fabric/peer# CORE_PEER_LOCALMSPID="Org1MSP"
root@f831468920b3:/opt/gopath/src/github.com/hyperledger/fabric/peer# 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.c
rt
root@f831468920b3:/opt/gopath/src/github.com/hyperledger/fabric/peer# peer channel join -b mychannel.block
2018-11-16 00:55:45.772 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2018-11-16 00:55:45.917 UTC [channelCmd] executeJoin -> INFO 002 Successfully submitted proposal to join channel
```

$ peer channel getinfo –c mychannel

```
root@f831468920b3:/opt/gopath/src/github.com/hyperledger/fabric/peer# peer channel getinfo -c mychannel
2018-11-16 00:57:40.757 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
Blockchain info: {"height":1,"currentBlockHash":"Qpw1UXOa6hxFSmhg9KKUJk7APgE11okZcoDac7Eml0c="}
```

: 현재 채널의 정보를 출력해보면 내용이 나옴(height라고 하는 이유는 peer들을 쌓기 때문)

$ peer channel update -o orderer.example.com:7050 -c mychannel –f ./channel-artifacts/Org1MSPanchors.tx --tls $CORE_PEER_TSL_ENABLED —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 peer의 정보를 update해주는 작업(peer0.org1, peer0.org2를 update)
(Org1MSPanchors.tx 파일이 필요한 정보를 가지고 있으므로 이를 anchor-peer와 연결해주는 작업)
(만약 에러가 난다면? 환경변수 설정이 잘못 되서 나는 것이기 때문에 위에 채널에 peer를 join해주는 4줄짜리 코드를 한 번 더 써서 지정해주고 다시 한 번 업데이트!!)

```
guru@guru:~/fabric-samples/first-network/channel-artifacts$ ll
total 36
drwxrwxr-x 2 guru guru  4096 Nov 15 18:05 ./
drwxrwxr-x 7 guru guru  4096 Nov 15 18:03 ../
-rw-r--r-- 1 guru guru   346 Nov 15 18:04 channel.tx
-rw-r--r-- 1 guru guru 12778 Nov 15 18:03 genesis.block
-rw-rw-r-- 1 guru guru     0 Nov 14 13:42 .gitkeep
-rw-r--r-- 1 guru guru   284 Nov 15 18:04 Org1MSPanchors.tx
-rw-r--r-- 1 guru guru   284 Nov 15 18:05 Org2MSPanchors.tx
```

```
root@bfae1121bd9d:/opt/gopath/src/github.com/hyperledger/fabric/peer# peer channel getinfo -c mychannel
2018-11-16 01:57:03.345 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
Blockchain info: {"height":3,"currentBlockHash":"E88XgSvJd0N4Fg3gHpbnrh2w+NNMGSwwN6+uME90+qw=","previousBlockHash":"sPLOrR8qvXeP4OXfBZACUdR52xbHZPhYC1GPJ0y5uIM="}
```

: 다시 해보면 anchor peer 두 개가 추가되어 height가 3이 됨을 볼 수 있음!!

$ sudo apt-get install tree

```
guru@guru:~/fabric-samples/chaincode/chaincode_example02/node$ rmate -p 52698 chaincode_example02.js
```

: 파일 구조를 tree형식으로 보기 위해 설치

: 이렇게 해서 chaincode_example02.js 파일을 열어보면 init, invoke, delete, query, main 등의 함수 안에 이체 정보(누구에게 +, 누구에게 -, 금액받기 등), set, get 등이 담겨있음.

```
guru@guru:~/fabric-samples/chaincode/chaincode_example02$ ll
total 20
drwxrwxr-x 5 guru guru 4096 Nov 14 13:42 ./
drwxrwxr-x 9 guru guru 4096 Nov 14 16:45 ../
drwxrwxr-x 2 guru guru 4096 Nov 14 13:42 go/
drwxrwxr-x 3 guru guru 4096 Nov 14 13:42 java/
drwxrwxr-x 2 guru guru 4096 Nov 14 13:42 node/
```

: chaincode 파일은 chaincode_example02에 다 들어있는데 총 3개 모드(nodejs, JAVA, golan)
: 각각의 디렉토리로 들어가서 $ rmate –p 52698 [해당경로] 로 확인가능!!(아래는 JAVA 코드)

```
public class SimpleChaincode extends ChaincodeBase {

    private static Log _logger = LogFactory.getLog(SimpleChaincode.class);

    @Override
    public Response init(ChaincodeStub stub) {…
    }

    @Override
    public Response invoke(ChaincodeStub stub) {…
    }

    private Response invoke(ChaincodeStub stub, List<String> args) {…
    }

    // Deletes an entity from state
    private Response delete(ChaincodeStub stub, List<String> args) {…
    }

    // query callback representing the query of a chaincode
    private Response query(ChaincodeStub stub, List<String> args) {…
    }

    public static void main(String[] args) {…
    }

}
```

```
root@bfae1121bd9d:/opt/gopath/src/github.com/chaincode# peer
Usage:
  peer [command]

Available Commands:
  chaincode   Operate a chaincode: install|instantiate|invoke|package|query|signpackage|upgrade|list.
  channel     Operate a channel: create|fetch|join|list|update|signconfigtx|getinfo.
  help        Help about any command
  logging     Log levels: getlevel|setlevel|revertlevels.
  node        Operate a peer node: start|status.
  version     Print fabric peer version.

Flags:
  -h, --help                    help for peer
      --logging-level string    Default logging level and overrides, see core.yaml for full syntax

Use "peer [command] --help" for more information about a command.
```

: docker로 들어가서 github.com/chaincode로 이동하면 ubuntu와 같은 파일들이 나와 있음.

: peer라는 명령어를 쳐보면 아래 사용할 수 있는 옵션들이 쭉 나옴(주로 사용하는 것은 네모박스 3가지)

$ peer chaincode install -n mycc_go -v 1.0 -p /opt/gopath/src/github.com/chaincode/chaincode_example02/go

```
root@bfae1121bd9d:/opt/gopath/src/github.com/chaincode# peer chaincode install -n mycc_go -v 1.0 -p github.com/chaincode/chaincode_example02/go
2018-11-16 02:45:58.491 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 001 Using default escc
2018-11-16 02:45:58.493 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 002 Using default vscc
2018-11-16 02:45:59.769 UTC [chaincodeCmd] install -> INFO 003 Installed remotely response:<status:200 payload:"OK" >
```

: peer에 chaincode를 install해주는 작업(이후 instance도 만들어 줄꺼!)

: chaincode는 같은 이름, 버전으로 올리면 ERROR가 발생하므로 반드시 이름 바꿔줘야 함!!

: go는 기본값이므로 그냥 작성하되, nodejs나 java의 경우 –l 옵션을 활용해 언어를 명시해주고 체인코드 이름과 마지막 경로를 바꿔주고 install 할 것!(마지막에 200 뜨면 성공)

$ peer chaincode list --installed

```
root@bfae1121bd9d:/opt/gopath/src/github.com/chaincode# peer chaincode list --installed
Get installed chaincodes on peer:
Name: mycc_go, Version: 1.0, Path: github.com/chaincode/chaincode_example02/go, Id: 8e46fdccc656ee04213f3f6adec74f15285a068f7e090add6262f0970a697bd8
Name: mycc_java, Version: 1.0, Path: /opt/gopath/src/github.com/chaincode/chaincode_example02/java, Id: d3c12621c4b531bb3e9c5599c3b563116d3a7ff99ec5d2e0f43de420ef664efe
Name: mycc_node, Version: 1.0, Path: /opt/gopath/src/github.com/chaincode/chaincode_example02/node, Id: aeea614a690ffee63bbfbcc4ea23c2120a1bd44a65ae5f4728c69b36e171cf6c
```

: 설치된 chaincode list를 출력하면 3개가 정상 설치 된 것을 알 수 있다.

$ peer chaincode instantiate –o orderer.example.com:7050 --tls $CORE_PEER_TSL_ENABLED —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_go –v 1.0 –c '{"Args":["init", "a", "100", "b", "200"]}' -P "AND('Org1MSP.member', 'Org2MSP.member')"

: 앞에서 install 한 golang 체인코드에 instance를 만들어주는 작업(-c는 생성자에 전달되는 초기화 값)

: 사전에 반드시!!! peer의 환경변수를 지정해주고 실행할 것(peer1~4까지 환경변수 써 준 이후 코드 작성)

```
root@745948a77b75:/opt/gopath/src/github.com/hyperledger/fabric/peer# peer chaincode instantiate -o orderer.example.com:7050 --tls $CORE_PEER_TSL_ENABLED --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_go -v 1.0 -c '{"Args":["init", "a", "100", "b", "200"]}' -P "AND ('Org1MSP.member', 'Org2MSP.member')"
2018-11-16 05:18:44.637 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 001 Using default escc
2018-11-16 05:18:44.637 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 002 Using default vscc
root@745948a77b75:/opt/gopath/src/github.com/hyperledger/fabric/peer#
root@745948a77b75:/opt/gopath/src/github.com/hyperledger/fabric/peer# peer chaincode list --installed
Get installed chaincodes on peer:
Name: mycc_go, Version: 1.0, Path: github.com/chaincode/chaincode_example02/go, Id: 8e46fdccc656ee0421373f6adec74f15285a0687fe090add6262f0970a697bd8
root@745948a77b75:/opt/gopath/src/github.com/hyperledger/fabric/peer#
root@745948a77b75:/opt/gopath/src/github.com/hyperledger/fabric/peer# peer chaincode list --instantiated -C mychannel
Get instantiated chaincodes on channel mychannel:
Name: mycc_go, Version: 1.0, Path: github.com/chaincode/chaincode_example02/go, Escc: escc, Vscc: vscc
```

: escc, vscc가 default로하여 instance가 생성되었으며, **$ peer chaincode list —installed**로 확인해보면 mycc_go 있음.

: **$ peer chaincode list —instantiated –C mychannel**로 확인해보면 mychannel에 생성된 instance가 보임.

: **ESCC(Endorser System Chaincode) : 그림에서 2~3번 과정 수행 시 호출되는 CC**

* 트랜잭션을 실행한 후 트랜잭션 실행 결과(트랜잭션 상태, 체인 코드 이벤트, reat/write set 등)를 포함하는 트랜잭션 응답 메시지에 서명을 넣기 위해 엔도싱 피어에 의해 호출
* invoke 함수는 5 - 7 개의 매개 변수를 수용 / 이 매개 변수는 Header, ChaincodeProposalPayload, ChaincodeID, Response, 시뮬레이션 결과, 이벤트 및 페이로드 가시성(payload visibility)

: **VSCC(Validator System Chaincode) : 그림에서 7번 과정 수행 시 호출되는 CC**

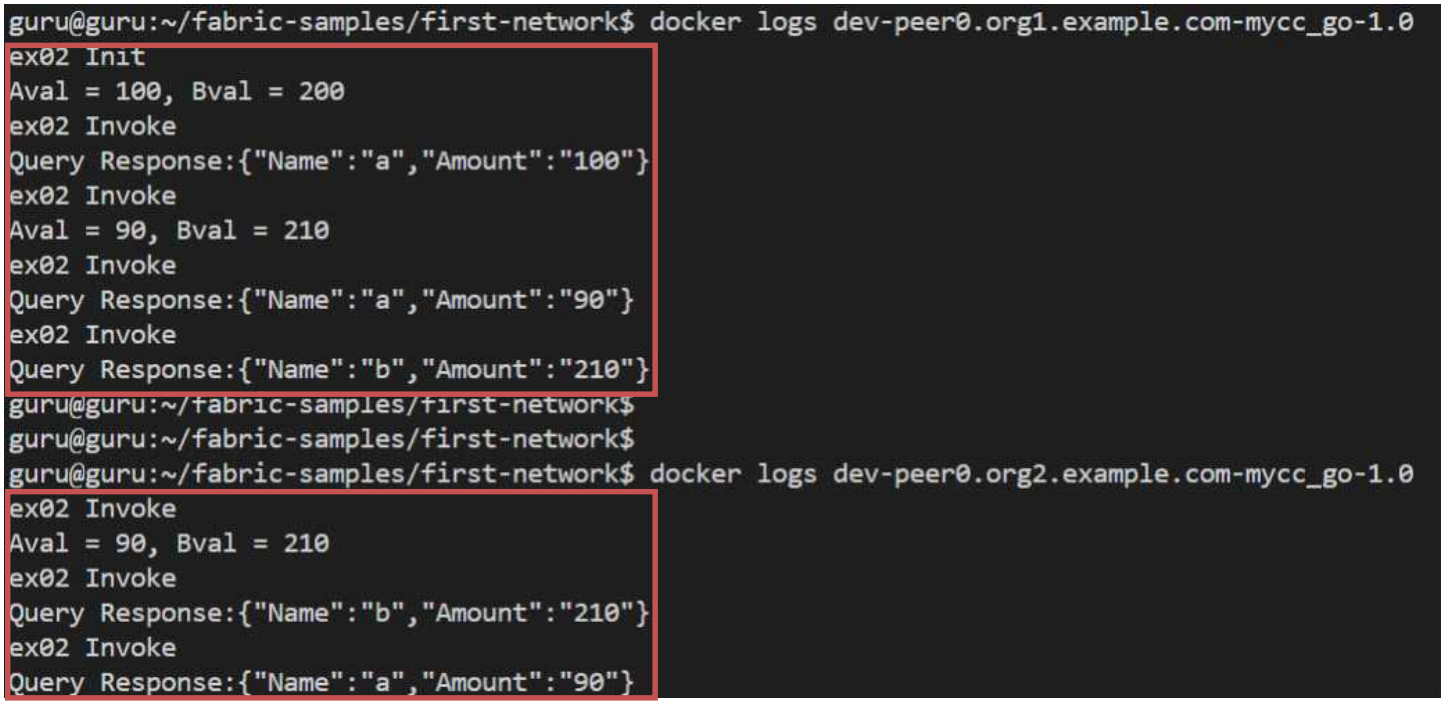
* committing 피어 (core/committer/txvalidator/validator.go)에 의해 호출되어 각 트랜잭션의 서명 세트를 체인 코드의 보증 정책에 대해 유효성을 검사

$ 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 mycc_go --peerAddresses peer0.org1.example.com:7051 —tlsRootCertFiles /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt --peerAddresses peer0.org2.example.com:7051 —tlsRootCertFiles /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt –c '{"Args":["invoke","a","b","10"]}'

```
root@745948a77b75:/opt/gopath/src/github.com/hyperledger/fabric/peer# 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 mycc_go --peerAddresses peer0.org1.example.com:7051 --tlsRootCertFiles /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt --peerAddresses peer0.org2.example.com:7051 --tlsRootCertFiles /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt -c '{"Args":["invoke","a","b","10"]}'
2018-11-16 05:48:44.858 UTC [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke successful. result: status:200
root@745948a77b75:/opt/gopath/src/github.com/hyperledger/fabric/peer#
root@745948a77b75:/opt/gopath/src/github.com/hyperledger/fabric/peer# peer chaincode query -C mychannel -n mycc_go -c '{"Args":["query","a"]}'
90
root@745948a77b75:/opt/gopath/src/github.com/hyperledger/fabric/peer# peer chaincode query -C mychannel -n mycc_go -c '{"Args":["query","b"]}'
210
```

: a가 b에게 10을 보내는 invoke 함수를 실행하면 status : 200과 함께 완료되었다는 메시지 띄움.

: **$ peer chaincode query –C mychannel –n mycc_go –c '{"Args":["query", "a"]}'와 "b"**로 확인해보면 각각 90과 210나와야함(금액 전송결과) → peer를 바꿔서 실행해 봐도 다 똑같음(broadcast 완료!!)



: 잘 전송이 되었는지 log정보를 통해 확인해보기(**$ docker logs dev-peer0.org1.example.com-mycc_go-1.0**)

: peer0에서 실행했기 때문에 Init부터 Invoke까지 모든 정보가 뜸!

: peer3에서 확인해보면 Invoke만 뜸!(앵커피어로 peer0에서 해당 정보를 전달받은 이후부터 출력)

3. 교재 실습!!

- p214(오픈이지/하이퍼레져 패브릭으로 구현하는 블록체인)에 나와 있는 https://github.com/IBM-Blockchain/marbles 과제 해보기!