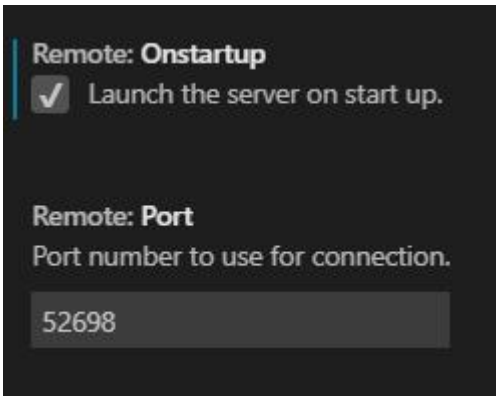
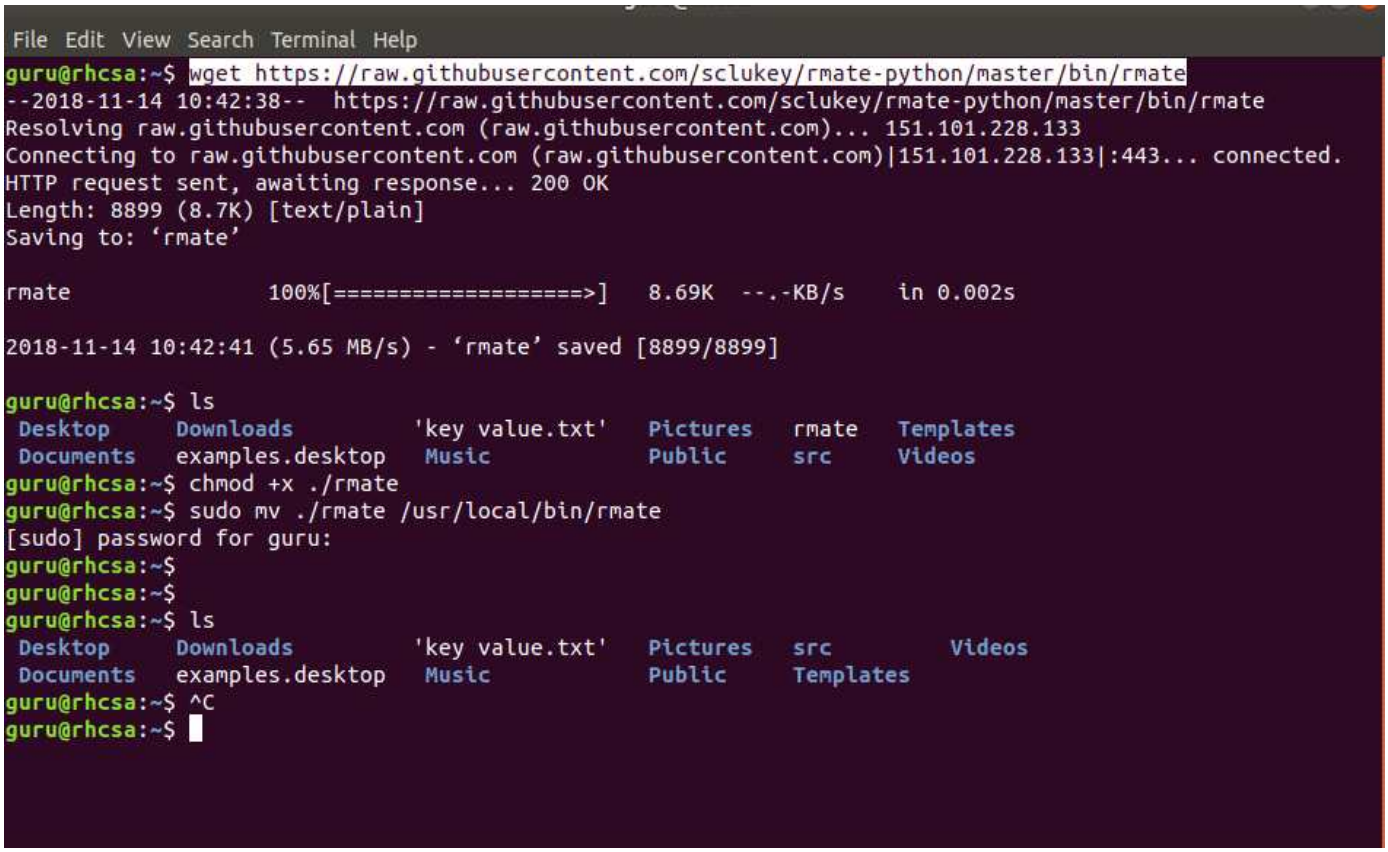
< 하이퍼레져 페브릭 환경설정 >

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)

- 내부 데이터 확인(10개가 있음)

```
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
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)