[2017]

HYPERLEDGER FABRIC v1.0 개발자 가이드

HYPERLEDGER FABRIC 을 이용한 애플리케이션 개발

MOO JE KONG

mooje.kong@gmail.com

1. Fabric SDK 로 트랜잭션 테스트 하기

Node.js 기반의 SDK 동작 테스트를 해봅니다. 앞서 블록체인 네트워크 테스트를 했던 fabric-samples 에서 fabcar 샘플을 통해 Fabric SDK 를 이용한 어플리케이션에서의 트랜잭션 테스트를 해보겠습니다. fabcar 샘플은 자동차에 대한 기본정보(모델명, 제조사, 색깔)와 소유자 정보를 블록체인으로 관리하는 샘플이며, 자동차 리스트 조회, 자동차 정보 생성, 소유자 변경등의 동작으로 테스트 해 볼 수 있습니다.

1.1. 테스트를 위한 블록체인 네트워크 실행하기

다음의 명령을 통해서 fabcar 디렉토리로 이동합니다.

```
cd ~/fabric-samples/fabcar
```

테스트를 위한 블록체인 네트워크를 실행하기 전에 이미 실행되고 있는 Hyperledger Fabric 컨테이너가 있다면 모두 중지 및 삭제 합니다.

```
docker stop $(docker ps -aq) && docker rm $(docker ps -qa)
```

그리고는 블록체인 네트워크를 실행합니다.

```
./startFabric.sh
```

그런 다음 다음 명령을 통해서 Fabric SDK 를 설치합니다. 이 명령어는 아래 그림과 같이 두개의 Fabric SDK(fabric-ca-client, fabric-client)를 설치합니다.

```
npm install
```

```
{
    "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": "^1.0.0",
        "fabric-client": "^1.0.0"
    },
    "author": "Anthony O'Dowd",
    "license": "Apache-2.0",
    "keywords": [
        "Hyperledger",
```

1.2. 트랜잭션 테스트

이번에는 query.js, invoke.js 를 통해서 조회인 데이터 입력 및 변경에 대한 테스트를 해보겠습니다.
먼저 다음 명령을 통해서 블록체인에서 현재 등록된 자동차 리스트를 검색해 보겠습니다.

```
node query.js
```

```
bcadmin@hlfv1:~/fabric-samples/fabcar$ node query
Create a client and set the wallet location
Set wallet path, and associate user  PeerAdmin  with application
Check user is enrolled, and set a query URL in the network
Make query
Assigning transaction_id:  c7a783a8405e9a96296a9ff223ef0661eb2269cb8a85c678ebf806a46eed7ea8
returned from query
Query result count =  1
Response is  [{"Key":"CAR0", "Record":{"colour":"blue","make":"Toyota","model":"Prius","owner":"Tomoko"}},{"Key":"CAR1", "Record":{
"colour":"red","make":"Ford","model":"Mustang","owner":"Brad"}},{"Key":"CAR2", "Record":{"colour":"green","make":"Hyundai","model":
"Tucson","owner":"Jin Soo"}},{"Key":"CAR3", "Record":{"colour":"yellow","make":"Volkswagen","model":"Passat","owner":"Max"}},{"Key"
:"CAR4", "Record":{"colour":"black","make":"Tesla","model":"S","owner":"Adriana"}},{"Key":"CAR5", "Record":{"colour":"purple","make
":"Peugeot","model":"205","owner":"Michel"}},{"Key":"CAR6", "Record":{"colour":"white","make":"Chery","model":"S22L","owner":"Aarav
"}},{"Key":"CAR7", "Record":{"colour":"violet","make":"Fiat","model":"Punto","owner":"Pari"}},{"Key":"CAR8", "Record":{"colour":"in
digo","make":"Tata","model":"Nano","owner":"Valeria"}},{"Key":"CAR9", "Record":{"colour":"brown","make":"Holden","model":"Barina","
owner":"Shotaro"}}]
```

다음으로는 새로운 자동차 정보를 등록해보겠습니다. invoke.js 파일을 열어서 다음과 같이 57,58 라인을 수정하고 다음의 명령을 실행합니다.

```
50      console.log("Assigning transaction_id: ", tx_id._transaction_id);
51      // createCar - requires 5 args, ex: args: ['CAR11', 'Honda', 'Accord', 'Black', 'Tom'],
52      // changeCarOwner - requires 2 args , ex: args: ['CAR10', 'Barry'],
53      // send proposal to endorser
54      var request = {
55          targets: targets,
56          chaincodeId: options.chaincode_id,
57          fcn: 'createCar',
58          args: ['CAR11', 'Honda', 'Accord', 'Black', 'Tom'],
59          chainId: options.channel_id,
60          txId: tx_id
61      };
62      return channel.sendTransactionProposal(request);
```

```
node invoke.js
```

```
bcadmin@hlfv1:~/fabric-samples/fabcar$ node invoke.js
Create a client and set the wallet location
Set wallet path, and associate user  PeerAdmin  with application
Check user is enrolled, and set a query URL in the network
Assigning transaction_id:  ec630a85602ee67e41f1ee4946975ae4b1480e1de1928a97750f0895cfe0667d
transaction proposal was good
Successfully sent Proposal and received ProposalResponse: Status - 200, message - "OK", metadata - "", endorsement signature: 0E!
info: [EventHub.js]: _connect - options {}
The transaction has been committed on peer localhost:7053
 event promise all complete and testing complete
Successfully sent transaction to the orderer.
```

정상적으로 처리되었으면 query.js 파일을 열어서 다음과 같이 51,52 라인을 수정하고 명령을 실행해서 정상적으로 CAR11 이 검색되는지 확인합니다.

```
45
46      // queryCar - requires 1 argument, ex: args: ['CAR4'],
47      // queryAllCars - requires no arguments , ex: args: [''],
48      const request = {
49          chaincodeId: options.chaincode_id,
50          txId: transaction_id,
51          fcn: 'queryCar',
52          args: ['CAR11']
53      };
54      return channel.queryByChaincode(request);
55  }).then((query_responses) => {
56      console.log("returned from query");
57      if (!query_responses.length) {
```

node query.js

```
bcadmin@hlfv1:~/fabric-samples/fabcar$ node query.js
Create a client and set the wallet location
Set wallet path, and associate user  PeerAdmin  with application
Check user is enrolled, and set a query URL in the network
Make query
Assigning transaction_id:  557e9ffd72124f085cf87a9d805735875f1c88fc0281f04f4fbefd5683ebcdb2
returned from query
Query result count =  1
Response is  {"colour":"Black","make":"Honda","model":"Accord","owner":"Tom"}
```

다음으로는 조금 전 등록했던 CAR11 에 대해서 소유권을 변경해보겠습니다.. invoke.js 파일의 57,58 라인을 다음과 같이 수정합니다. 기존에 "Tom"의 소유였던 CAR11 을 "Barry"로 변경하는 내용입니다. 수정 후 다음의 명령을 실행하여 블록체인에 트랜잭션을 실행합니다.

```
49      tx_id = client.newTransactionID();
50      console.log("Assigning transaction_id: ", tx_id._tr
51      // createCar - requires 5 args, ex: args: ['CAR11',
52      // changeCarOwner - requires 2 args , ex: args: ['C
53      // send proposal to endorser
54      var request = {
55          targets: targets,
56          chaincodeId: options.chaincode_id,
57          fcn: 'changeCarOwner',
58          args: ['CAR11', 'Barry'],
59          chainId: options.channel_id,
60          txId: tx_id
61      };
62      return channel.sendTransactionProposal(request);
63  }).then((results) => {
```

```
node invoke.js
```

```
bcadmin@hlfv1:~/fabric-samples/fabcar$ node invoke.js
Create a client and set the wallet location
Set wallet path, and associate user  PeerAdmin  with application
Check user is enrolled, and set a query URL in the network
Assigning transaction_id:  43e9c78d0d627122d82aa1258d96cc75f069a82e79803ba09c65b2143a0bd3e5
transaction proposal was good
Successfully sent Proposal and received ProposalResponse: Status - 200, message - "OK", metadata - "", endorsement signature: 0D |
;
info: [EventHub.js]: _connect - options {}
The transaction has been committed on peer localhost:7053
 event promise all complete and testing complete
Successfully sent transaction to the orderer.
```

다음과 같이 정상적으로 처리되었으면 다시 쿼리를 해서 정상적으로 소유권 변경이 되었는지 확인해봅니다.

```
node query.js
```

```
bcadmin@hlfv1:~/fabric-samples/fabcar$ node query.js
Create a client and set the wallet location
Set wallet path, and associate user  PeerAdmin  with application
Check user is enrolled, and set a query URL in the network
Make query
Assigning transaction_id:  361099d4b801e12d82fd3348dc938e2b26cd636f59e2424eb7159aa0eb1b8704
returned from query
Query result count =  1
Response is  {"colour":"Black","make":"Honda","model":"Accord","owner":"Barry"}
```

1.3. 이벤트 리스너 사용하기

다음의 경로로 이동하여 이벤트 리스너 코드를 빌드합니다.

```
cd $GOPATH/src/github.com/hyperledger/fabric/examples/events/block-listener/
go build
```

빌드 과정에서 ltdl.h 와 관련한 에러가 날 경우에 다음의 명령을 통해 시스템 라이브러리를 설치한 후 다시 빌드합니다.

```
sudo apt-get install -y libltdl-dev
```

빌드가 완료되었으면 다음의 명령을 통해서 이벤트 리스너를 실행합니다.

```
./block-listener -events-address=127.0.0.1:7053 -events-mspdir=/home/bcadmin/fabric-samples/basic-network/crypto-config/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp -events-mspid=Org1MSP
```

Invoke 트랜잭션을 발생시켜 이벤트 발생을 모니터링 합니다.

2. 스마트 컨트랙트 (체인코드) 개발

이번 장에서는 개발모드에서 체인코드를 개발하고 테스트 하는 방법에 대해서 설명합니다. fabric-samples/chaincode-docker-devmode 에서 테스트합니다.

샘플 파일이 있는 위치로 이동하고 현재 실행중인 컨테이너가 있다면 모두 중지합니다.

```
cd ~/fabric-samples/chaincode-docker-devmode
docker stop $(docker ps -qa) && docker rm $(docker ps -qa)
```

블록체인 네트워크를 실행하고 채널 생성을 합니다.

```
docker-compose -f docker-compose-simple.yaml up –d
```

다음으로는 체인코드를 빌드해서 실행합니다. 빌드를 하기 위해서는 체인코드 빌드를 위한 환경을 제공하고 있는 hyperledger/fabric-ccenv 컨테이너에서 빌드를 하면되고 현재 실행되고 있는 컨테이너명은 chaincode 입니다.
다음의 명령을 통해서 chaincode 컨테이너에 접속하고 sacc 체인코드를 빌드합니다.

```
docker exec -it chaincode bash
cd sacc
go build
```

정상적으로 빌드를 완료하였으면 다음의 명령을 통해서 체인코드를 실행합니다.
CORE_PEER_ADDRESS 는 체인코드가 붙어서 통신해야하는 Peer 의 접속 정보를 입력해주면 되고,
CORE_CHAINCODE_ID_NAME 은 체인코드의 이름과 버전입니다.
현재는 체인코드의 프로세스만 실행을 시킨 상태이고 채널에 등록이 된 상태는 아닙니다.

```
CORE_PEER_ADDRESS=peer:7051 CORE_CHAINCODE_ID_NAME=mycc:0 ./sacc
```

그리고는 채널에 체인코드를 등록하고 초기화 시키기 위해서 **새로운 터미널을 실행하여** cli 컨테이너에 접속합니다.

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

다음의 명령을 통해서 체인코드를 채널에 등록하고 초기화합니다.

```
peer chaincode install -p chaincodedev/chaincode/sacc -n mycc -v 0
peer chaincode instantiate -n mycc -v 0 -c '{"Args":["a","10"]}' -C myc
```

다음 코드를 통해서 트랜잭션을 실행해서 정상 동작을 확인합니다.

```
peer chaincode invoke -n mycc -c '{"Args":["set", "a", "20"]}' -C myc
peer chaincode query -n mycc -c '{"Args":["query","a"]}' -C myc
```

수정된 체인코드의 빌드 → 채널의 등록(install, instantiate) → 체인코드 호출 (invoke, query) 등을 반복해서 작업합니다.

3. Hyperledger Composer 를 이용한 비즈니스 네트워크 개발

3.1. Hyperledger Composer 설치

다음의 명령에 따라 Hyperledger Composer 를 설치합니다.

```
npm install -g composer-cli
npm install -g generator-hyperledger-composer
npm install -g composer-rest-server
npm install -g yo

# 로컬에 Playground 를 사용하기 위해서는 다음을 설치합니다.
npm install -g composer-playground
```

3.2. Hyperledger Fabric 시작

Hyperledger Composer 를 사용하기 위해 우선 Hyperledger Fabric 을 시작합니다.
다음의 명령을 통해 테스트를 위한 디렉토리를 만들고 필요한 파일을 다운로드 받습니다.

```
mkdir ~/fabric-tools && cd ~/fabric-tools
curl -O https://raw.githubusercontent.com/hyperledger/composer-tools/master/packages/fabric-dev-servers/fabric-dev-servers.tar.gz
tar xvz fabric-dev-servers.tar.gz
```

현재 Hyperledger Composer 는 Hyperledger Fabric v0.6 과 v1.0-alpha 에서 테스트 가능합니다. 다음의 명령어를 통해 환경변수를 설정합니다.

```
export FABRIC_VERSION=hlfv1
```

만약, 처음 시작하는 분은 도커 이미지를 다운로드하기 위해 ./downloadFabric.sh 를 실행하여야 하고 도커 이미지가 있는 경우에는 다운로드 스크립트는 실행하지 않아도 됩니다.

```
cd ~/fabric-tools
./downloadFabric.sh
./startFabric.sh
./createComposerProfile.sh
```

모든 테스트가 완료되고 실행환경을 중지하여야 할 때는 다음의 명령어를 실행하시기 바랍니다.

```
./stopFabric.sh
./teardownFabric.sh
```

3.3. Hyperledger Composer 를 이용한 비즈니스 네트워크 생성

다음 명령을 통해 Hyperledger Composer 를 위한 샘플 비즈니스 네트워크 정의 파일을 다운로드 받습니다.

```
cd ~/fabric-tools
git clone https://github.com/hyperledger/composer-sample-networks.git
cp -r composer-sample-networks/packages/basic-sample-network/ ./my-network
cd my-network
```

packages.json 파일을 다음과 같이 수정합니다.

```
{
    "name": "my-network",
    "version": "0.0.1",
    "description": "My very first Hyperledger Composer Network",
    "scripts": {
        "prepublish": "mkdirp ./dist && composer archive create --sourceType dir --sourceName . -a ./dist/my-network.bna",
        "pretest": "npm run lint",
        "lint": "eslint .",
        "postlint": "npm run licchk",
        "licchk": "license-check",
        "postlicchk": "npm run doc",
        "doc": "jsdoc --pedantic --recurse -c jsdoc.conf",
        "test": "mocha --recursive -t 4000"
    },
...
}
```

도메인 모델 파일을 수정합니다. 도메인 모델 파일은 models/sample.cto 파일입니다.

```
/**
 * My commodity trading network
 */
namespace org.acme.mynetwork
asset Commodity identified by tradingSymbol {
    o String tradingSymbol
    o String description
    o String mainExchange
    o Double quantity
    --> Trader owner
}
participant Trader identified by tradeId {
    o String tradeId
    o String firstName
    o String lastName
}
transaction Trade {
```

```
    --> Commodity commodity
    --> Trader newOwner
}
```

트랜잭션 프로세스 함수를 수정합니다. 트랜잭션 처리 함수는 lib/sample.js 파일입니다. 파일의 전체 내용을 다음의 내용으로 교체합니다.

```
/*
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

/**
 * Track the trade of a commodity from one trader to another
 * @param {org.acme.mynetwork.Trade} trade - the trade to be processed
 * @transaction
 */
function tradeCommodity(trade) {
    trade.commodity.owner = trade.newOwner;
    return getAssetRegistry('org.acme.mynetwork.Commodity')
        .then(function (assetRegistry) {
            return assetRegistry.update(trade.commodity);
        });
}
```

다음으로 Access Control Rules 를 업데이트 합니다. 파일은 permissions.acl 입니다. 전체 내용을 교체합니다.

```
/**
 * Access control rules for mynetwork
 */
rule Default {
    description: "Allow all participants access to all resources"
    participant: "ANY"
    operation: ALL
    resource: "org.acme.mynetwork.*"
    action: ALLOW
```

```
}
```

여기까지 필요한 파일을 업데이트 하였으면 Business Network Archive 를 생성합니다.

```
# 현재 작업 디렉토리가 ~/fabric-tools/my-network 가 맞는지 확인합니다.
npm install
```

위의 명령이 성공적으로 완료되면 dist 디렉토리에 my-network.bna 파일이 생성됩니다.

3.4. Hyperledger Fabric 런타임에 비즈니스 네트워크 아카이브 디플로이

위의 과정을 정상적으로 완료하였으면 이제 아카이브 파일을 Hyperledger Fabric 런타임에 디플로이 할 차례입니다. 다음의 명령을 통해 디플로이합니다.

```
composer network deploy -a dist/my-network.bna -p hlfv1 -i PeerAdmin -s anything
```

다음과 같은 메세지가 나오면 정상적으로 디플로이가 완료된 것입니다.

```
Deploying business network from archive: my-network.bna
Business network definition:
Identifier: my-network@0.0.1
Description: My very first Hyperledger Composer Network

✔ Deploying business network definition. This may take a minute...


Command succeeded
```

다음 명령을 통해 네트워크 동작을 확인합니다.

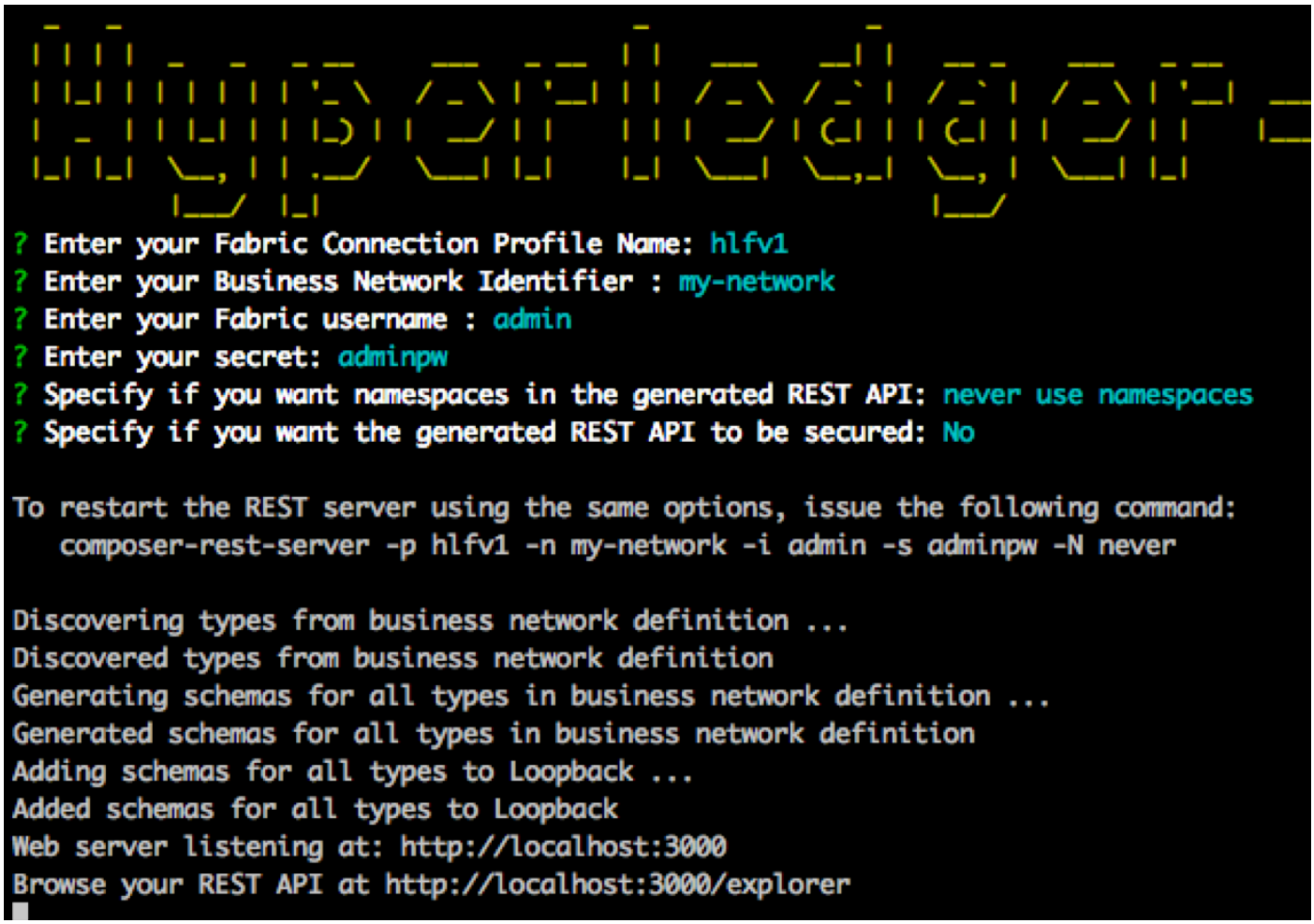
```
composer network ping -n my-network -p hlfv1 –i admin -s adminpw
```

3.5. REST API 생성 및 테스트

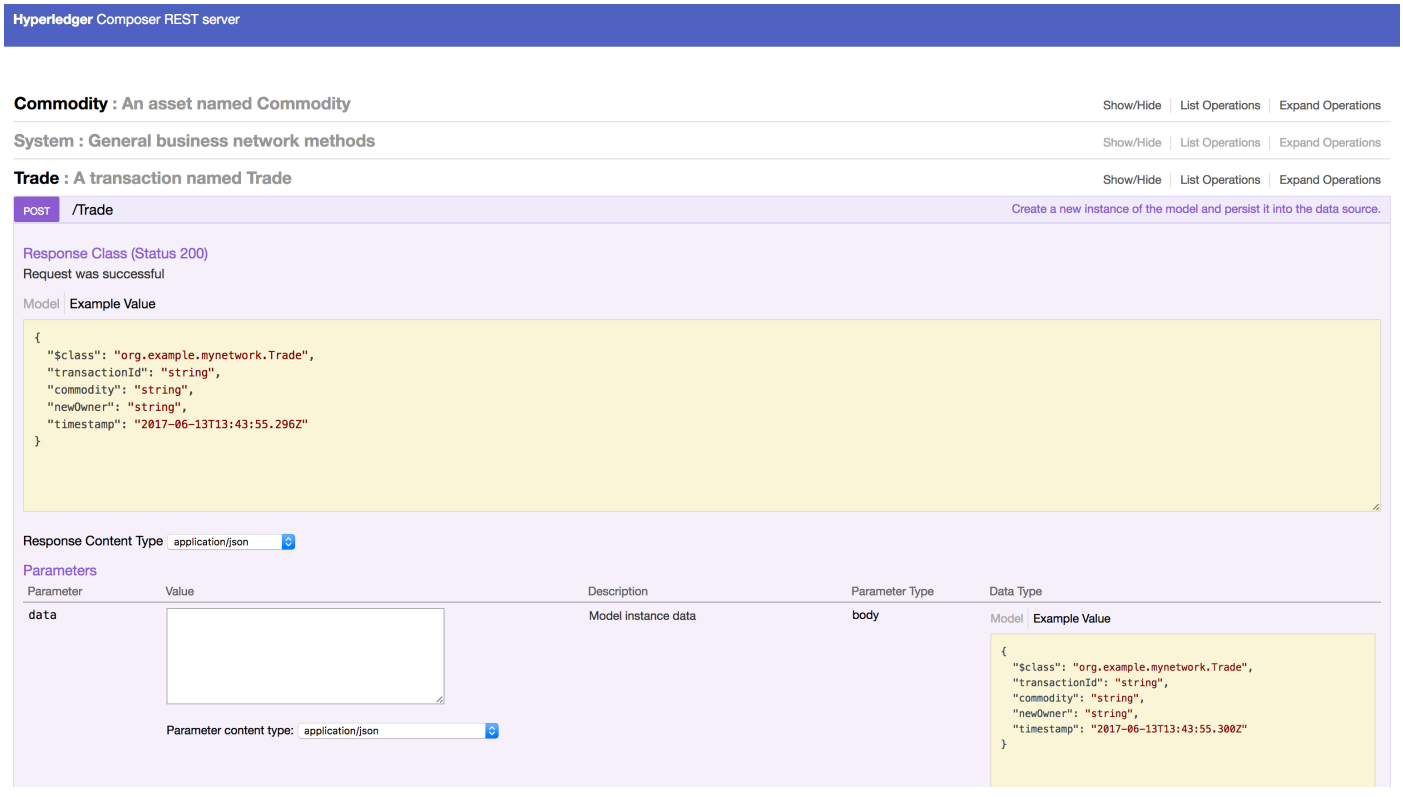
다음의 명령을 통해서 REST 서버를 실행합니다.

```
composer-rest-server
```

다음의 화면과 같이 선택합니다. 마지막 메세지와 같이 정상적으로 구동되었으면 http://localhost:3000/explorer 로 접속합니다. (VM 환경에서 실행 할 경우 호스트네임을 VM 의 IP 주소로 변경해서 접속합니다.)



다음과 같은 REST API 화면을 보실 수 있으며, API 테스트를 해보시기 바랍니다.



3.6. 스켈레톤 웹 어플리케이션 생성

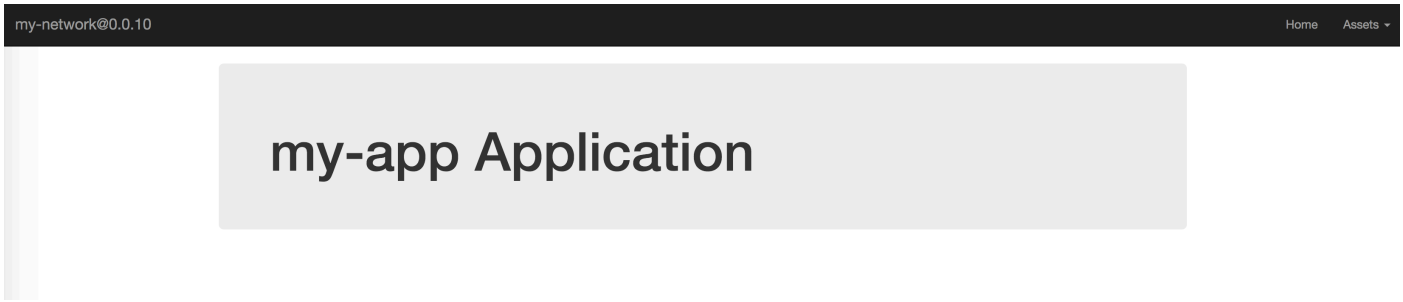
REST API 서버는 Ctrl+C 로 종료합니다. 다음의 명령어를 통해 웹 어플리케이션을 생성합니다.

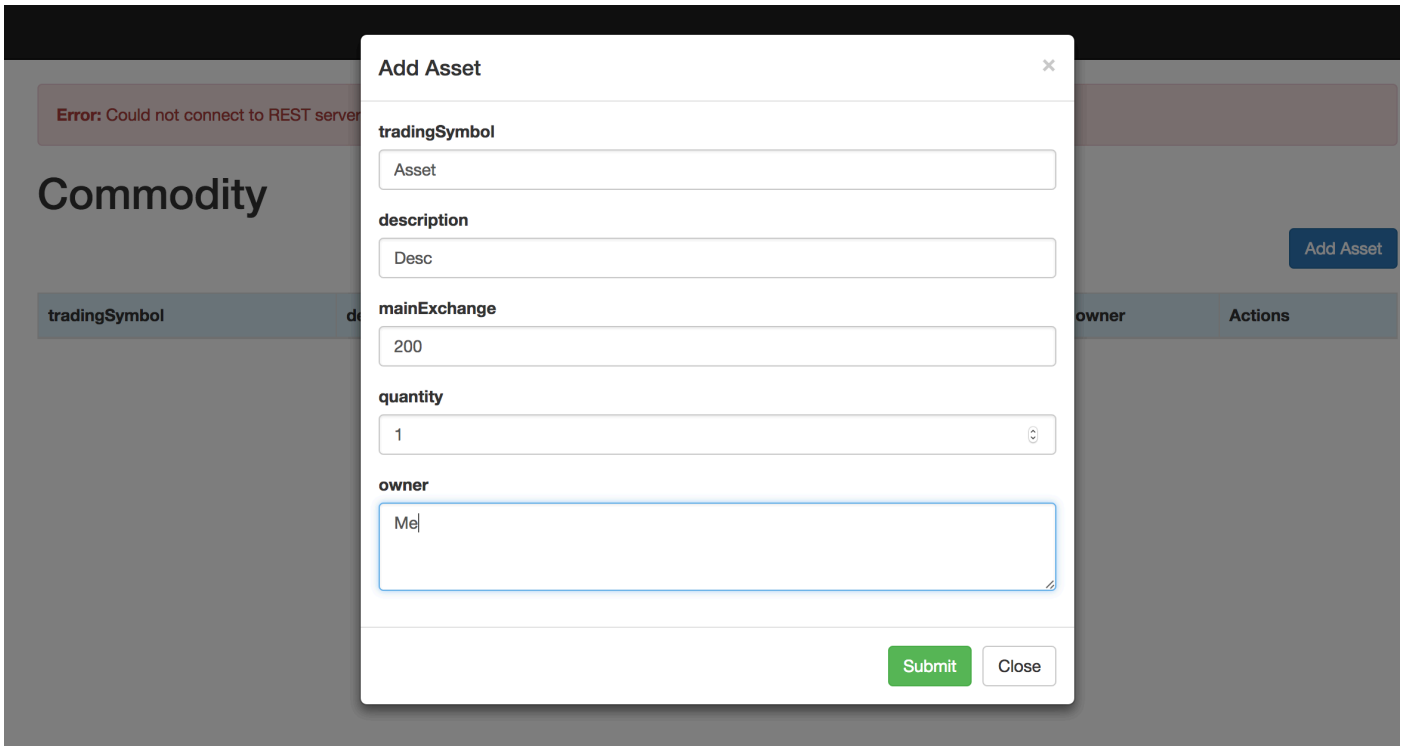
```
cd ~/fabric-tools/
yo hyperledger-composer
```

다음과 같은 과정으로 진행합니다.

```
bcadmin@fabric01:~/fabric-tools$ yo hyperledger-composer
Welcome to the Hyperledger Composer Skeleton Application Generator
? Please select the type of Application: Angular 2 Application
You can run this generator using: 'yo hyperledger-composer:angular'
Welcome to the Hyperledger Composer Angular 2 skeleton application generator
? Do you want to connect to a running Business Network? Yes
? What is the name of the application you wish to generate?: my-app
? Description of the application: Commodities Trading App
? Author name: mjkong
? Author email: mjkong@kr.ibm.com
? What is the Business Network Identifier?: my-network
? What is the Connection Profile to use? hlfv1
? Enrollment id: admin
? Enrollment Secret: adminpw
? Do you want to generate a new REST API or connect to an existing REST API?:  Generate a new REST API
? What port number should the generated REST server run on?: 3000
? Should namespaces be used in the generated REST API:  Never use namespaces
About to connect to a running business network
Connected to: my-network
Created application!
Completed generation process
```

브라우저에서 http://localhost:4200 으로 접속합니다.




Error: Could not connect to REST server
Commodity
Add Asset
tradingSymbol
Actions
Add Asset
tradingSymbol
Asset
description
Desc
mainExchange
200
quantity
1
owner
Me
Submit
Close