HLF-generator

박재훈

한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB
https://crypto.modoo.at

설명

- 하이퍼레저 패브릭 프로젝트를 쉽게 생성할 수 있는 제너레이터

- Hyperledger Fabric 2.2 LTS

- Python 3.7
  - PyYAML

작동 방식

- 조직, 피어 등의 이름과 그 수를 입력 받아 구동에 필요한 파일들 생성

- 체인코드의 경우 기본적인 파일만 제공하며 이후 프로그래머가 알아서 짜는 방식

노드 구성

- 하이퍼레저 패브릭은 조직, 피어, 채널 등의 단위로 구성되어 있음

- 각각에 대한 클래스를 정의하고 그것을 바탕으로 진행

하이퍼레저 패브릭 쉘스크립트 내부 설정

- network/scripts/utils.sh

```
_org1_p0_ch0=(org1 Org1 peer0 Admin 7051 mychannel)
_org1_p1_ch0=(org1 Org1 peer1 Admin 8051 mychannel)
_org2_p0_ch0=(org2 Org2 peer0 Admin 9051 mychannel)
_org2_p1_ch0=(org2 Org2 peer1 Admin 10051 mychannel2)
_org3_p0_ch0=(org3 Org3 peer0 Admin 11051 mychannel2)
_org3_p1_ch0=(org3 Org3 peer1 Admin 12051 mychannel2)
```

- 각 피어의 채널을 기준으로 객체 설정
- 하나의 채널에 대해
  - 객체(배열)명 : _{org-addr}_p{peer order}_ch{channel order}
  - 객체 내용 : org-addr, org-name, peer-name, admin-name, peer-port, channel-name
- 이와 같은 방식으로 제너레이터에서도 클래스 설정

제너레이터에서의 클래스 설정

- 하이퍼레저 패브릭 내부에서의 변수에 대한 설정을 파이썬에서 진행

- 5개의 클래스로 구성

- Orderer
- OrdererItem
- Organ
- Peer
- FabChannel

class Orderer

- 오더러를 정의하는 클래스
- string msp : 오더러의 MSP 이름 (ex. OrdererMSP)
- int caport : 오더러의 CA 포트
- List[OrdererItem] items : 오더러를 구성하는 아이템들

class OrdererItem

- 오더러가 여러 포트로 구성되어 있을 경우, 각각의 아이템들에 대한 정의를 내려주는 클래스
- string addr : 주소에서 쓰일 아이템 이름
- string name : 표현될 아이템 이름
- int port : 포트번호 (defulat 7050, 1000씩 늘어나는 규칙)
- Orderer orderer : 자신이 속한 오더러

class Organ

- 조직을 정의

- string addr : organization name that can be represented in its address (ex. org1)
- string name : organization name (ex. Org1)
- string msp : organization msp name (ex. Org1MSP)
- int caport : CA port of the organization (starts from 7054)
- string admin : admin name of the organization (default ‘admin’)
- string adminpw : admin password of the organization (default ‘adminpw’)
- List[Peer] peers : peers that belongs to the organization

class Peer

- 피어를 정의

- Organ org : an organization that the peer belongs to
- string name : name of the peer (ex. peer0)
- string port : port of the peer (ex. 7051)
- string dbport : port of the peer’s CouchDB (ex. 5984)
- List[FabChannel] channels : channel list of the peer

class FabChannel

- 채널에 대한 정의
- string channel : name of the channel
- string profile : name of the channel profile
- string consortium : name of the consortium

default values

- 필드를 비우면 default value가 입력되는 식

- service name (like ‘example’ of ‘peer0.org1.example.com’) : example
- project name : Example
- project creator : John Doe
- consortium : Consortium
- network profile : TestNetworkProfile

- orderer port : 7050 (, 8050, 9050, ...)
- peer port : 7051 (, 8051, 9051, ...)
- peer db port : 5984 (, 6984, 7984, ...)
- ca port : 7054 (, 8054, 9054, ...)

입력 방식

let o := number of organizations

for i in o; then

input informations of the organization i

let p := number of peers of the organization i

for j in p; then

input informations of the peer j

let c := channels that the peer j belongs to (e.g. ‘channel1 channel2’)

for k in c; then

input informations of the channel k

append itself to the peer j

append itself to the organ i

append itself to the organizations list

입력 방식

let orderer
input informations of the orderer

let o := number of orderer items
for i in o; then
    input informations of the item i
    append itself to the orderer

디렉토리 구조

- exports
  - *exported projects*
- history
  - *previous commands*
- src
  - *python files*
- template
  - chaincode
    - codes
      - *code templates for chaincode*
    - packages
  - invoke
    - *invoke templates for invoking*
  - network
    - *network templates for the network*

체인코드

- 타입스크립트

- 기본적으로 3개 파일로 구성
  - TEMPLATE_CC.ts
  - index.ts
  - utils.ts

- TEMPLATE_CC.ts의 파일명은 프로젝트 생성 시 변경됨

체인코드 - TEMPLATE_CC.ts

- initLedger, getAll, get, create, update, delete 함수의 튜토리얼을 포함
- initLedger는 비어있는 상태
- 프로그래머가 새로운 모델을 만들어 함수들을 수정/추가해서 사용하는 식

```
async create(ctx: Context, value: any) {
    const timestamp = ctx.stub.getTxTimestamp().seconds.low;
    const data = {
        key: `T-${timestamp}`,
        value: value
    };
    await ctx.stub.putState(data.key, utils.stateValue(data));
}
```

체인코드 - index.ts

```
import { {{TEMPLATE_CC}} } from './{{CC_FILENAME}}';
export { {{TEMPLATE_CC}} } from './{{CC_FILENAME}}';

export const contracts: any[] = [ {{TEMPLATE_CC}} ];
```

체인코드 - utils.ts

```
export function stateValue(value: any) {
    return Buffer.from(JSON.stringify(value));
}

export function toItem(item: Uint8Array) {
    return JSON.parse(item.toString());
}
```

invoke

- 체인코드를 실행할 때 사용하는 파트

- 타입스크립트

- 5개의 실행파일
  - enrollAdmin.ts
  - registerUser.ts
  - invoke.ts
  - server.ts
  - utils.ts

- 1개의 데이터 파일
  - data.json

invoke

- invoke 실행 시 enrollAdmin, registerUser 사전 실행 필요
  - 어드민과 유저를 등록함으로써 지갑 생성

- invoke.ts : 인자로 입력받은 파라미터를 바탕으로 query, invoke 실행
  - 파라미터 1번째 : 실행할 체인코드 함수명
  - 파라미터 2+번째 : 체인코드 함수에 대한 파라미터
  - query(evaluateTransaction) : DB 조회 (값 변경 X)
  - invoke(submitTransaction) : DB 값 변경

invoke - server.ts

- express로 돌아가는 백엔드 서버
- 포트 : 3000 (localhost:3000)
- RESTful API를 이용하여 체인코드 함수들에 연결하는 튜토리얼 기본 제공
  - 경로는 기본적으로 /api/samples
- doTransaction 함수를 이용하여 네트워크 연결 및 해제 과정 생략
  - doTransaction에 콜백 작성

```
app.get('/api/samples', async (req, res, next) => {
    doTransaction(req, res, next, async contract => {

        const result = await contract.evaluateTransaction('getAll');
        const obj = utils.bufferToObject(result);
        res.json(obj);
    });
});
```

invoke - data.json

- 프로젝트 전체 정보를 포함
  - 서비스명, 네트워크 프로필, 체인코드명, 오더러, 조직

- invoke에서는 특정 피어를 이용해서 query, invoke 작업을 수행
  - 어떤 피어를 바탕으로 작업을 수행해야 할지 모름
  - 프로젝트 생성 과정에서 어떤 피어를 사용할지를 delegate로써 설정

- utils.ts에서 data.json을 불러와서 내부 변수 설정

```
"delegate": {
    "addr": "org1",
    "name": "Org1",
    "msp": "Org1MSP",
    "caport": 7054,
    "admin": "admin",
    "adminpw": "adminpw",
    "peer": {
        "name": "peer0",
        "port": 7051,
        "dbport": 5984,
        "channel": {
            "channel": "mychannel",
            "profile": "MyChannelProfile",
            "consortium": "Consortium"
        }
    },
    "chaincode": "Example"
}
```

Q & A