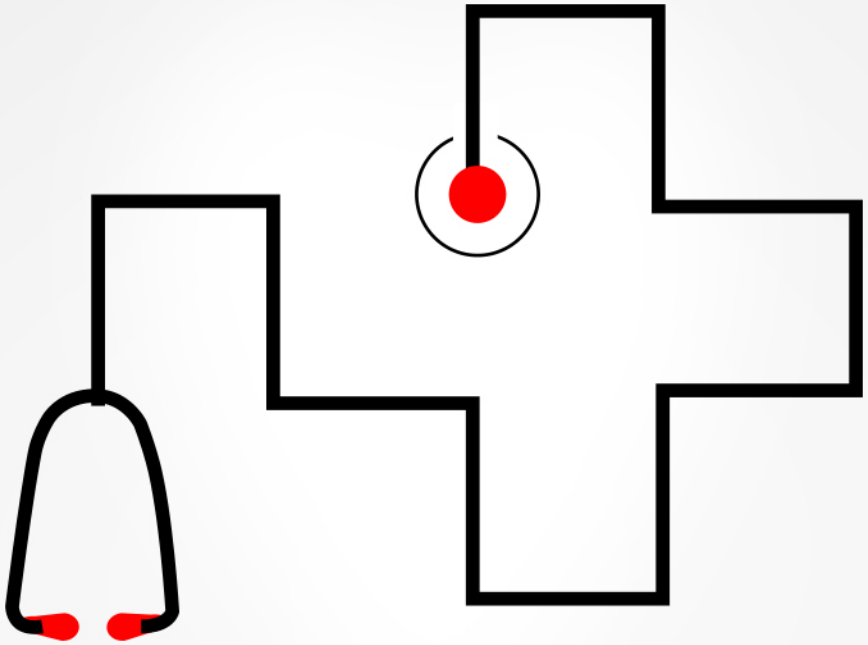


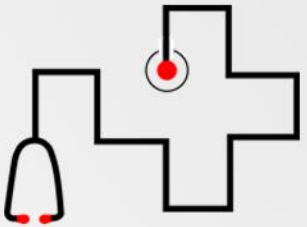
졸업 프로젝트2 T7

1st Implementation Demo

컴퓨터공학과

201511272 양재민, 201411295 이상훈, 201511295 조범석

0. 목차

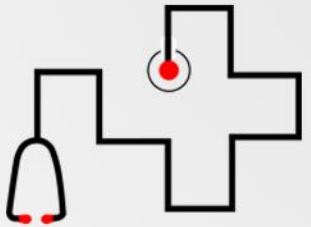


1. 블록체인 네트워크 환경 구성

2. IPFS 네트워크 환경 구성

3. 딥러닝 모델 구동

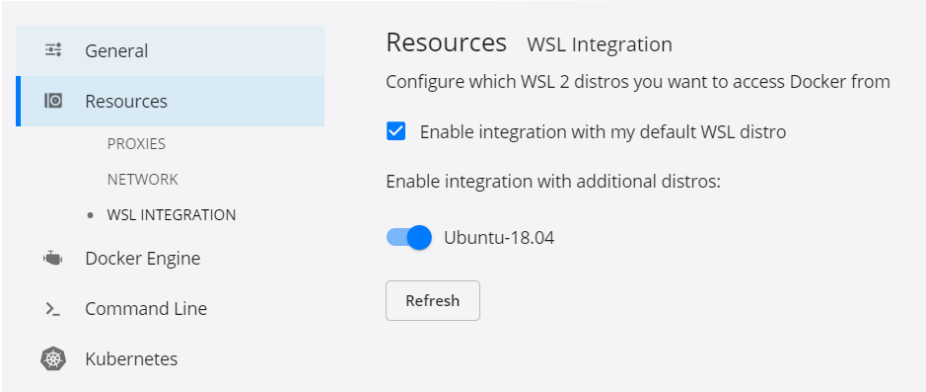
1. 블록체인 네트워크 환경 구성



0. Prerequisites - Hyperledger Fabric 구성에 필요한 필수요소 설치

(OS: Linux in Window 10 (WSL2), Docker For Windows 가 설치 되어 있어야 하고, WSL2 에서 사용가능해야 함.)

Docker for Windows 설정참고 ▶

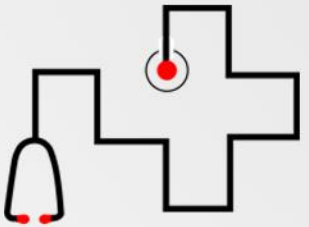



```
~ cd test_fabric
~/test_fabric master chmod 777 requisitesInstall.sh
~/test_fabric master ./requisitesInstall.sh
```

'test_fabric' 폴더로 이동해 네트워크 실행에 필요한 프로그램을 설치하는 스크립트 파일을 실행한다. (두번째 명령어는 쉘 스크립트의 권한을 수정)

requisitesInstall.sh로 설치할 수 있는 목록 - git, curl, docker, docker-compose 이외에 Node.js, npm, Go 설치 후 환경 변수까지 설정해준다.

1. 블록체인 네트워크 환경 구성



1. 블록체인 네트워크 구동 (실행 및 출력)

'test_fabric' 디렉토리에서 'test_medichain' 디렉토리로 이동 후,
'network_starter.sh' 스크립트 파일 실행

```
~/test_fabric  master  cd test-medichain
~/test_fabric/test-medichain  master  ./network-starter.sh
```

```
############################################################
##### Generate certificates using Fabric CA's ############
############################################################
Creating network "net_test" with the default driver
Creating ca_org1    ... done
Creating ca_org2    ... done
Creating ca_orderer ... done
############################################################
############ Create Org1 Identities ######################
############################################################
```

```
Creating couchdb1             ... done
Creating couchdb0             ... done
Creating orderer.example.com  ... done
Creating peer0.org2.example.com ... done
Creating peer0.org1.example.com ... done
```

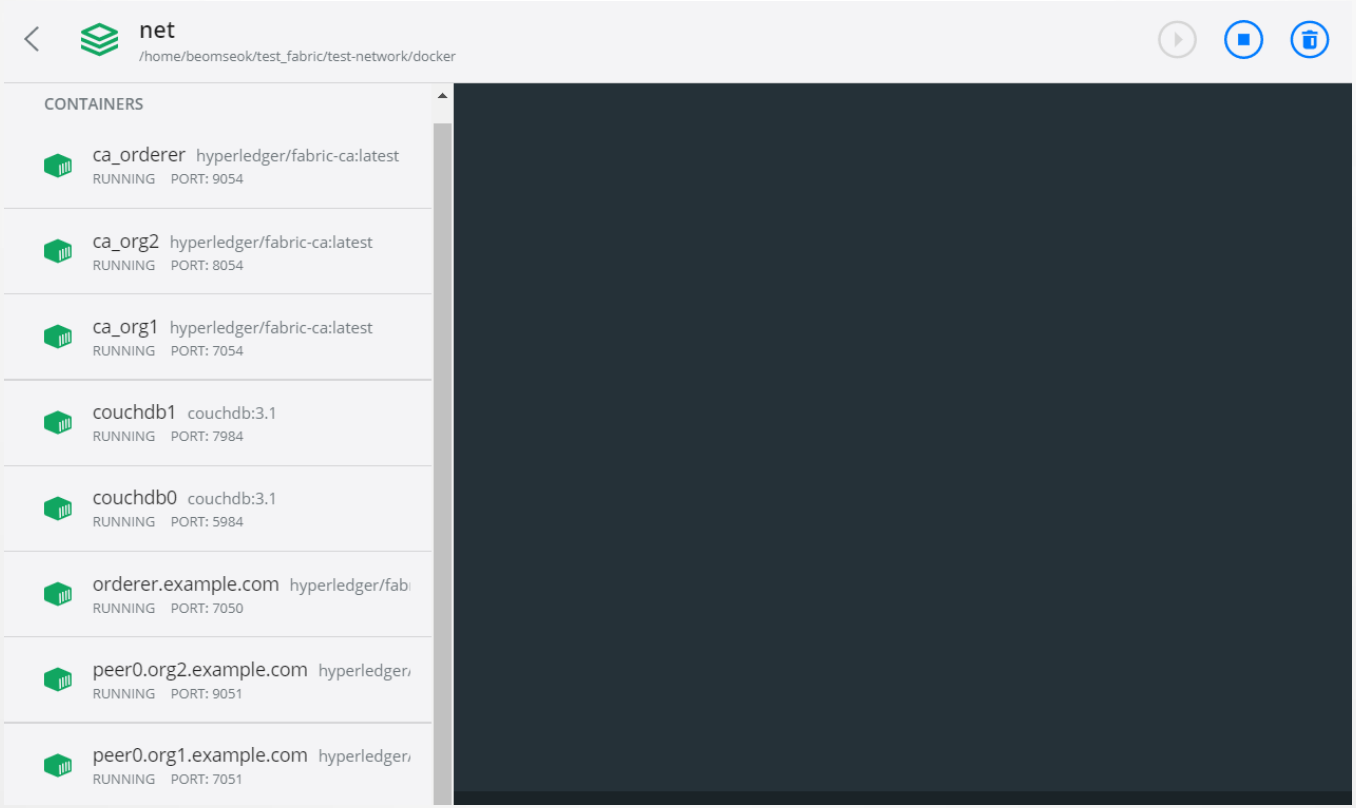
```
==================== Channel 'mychannel' created ====================
```

CA (Certificate Authority), CouchDB, Organization1(Node), 2(Node),
Orderer(Node) 를 Docker Container로 구성하고,
'mychannel' 채널을 만들어 블록체인 원장 하나를 만든다

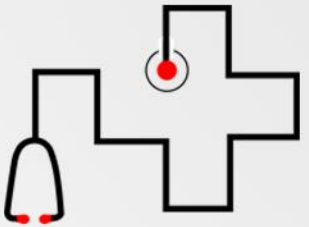
1. 블록체인 네트워크 환경 구성

1. 블록체인 네트워크 구동 (결과 화면)

Org1(=Doctor), Org2(=Patient), Orderer(=트랜잭션 순서) 관리하는 Organization
각 Organization 은 Node를 하나씩 가지고 있다.



1. 블록체인 네트워크 환경 구성



2. Hyperledger Fabric, Hyperledger Explorer 연결

Docker를 활용한 Fabric, Explorer 연결

- 이미지를 Dockerhub에서 받아서, 예시 파일을 설치해준다.
- 메디체인 네트워크를 실행 후 Organizations 정보들을 가져와 복사 후 실행한다.

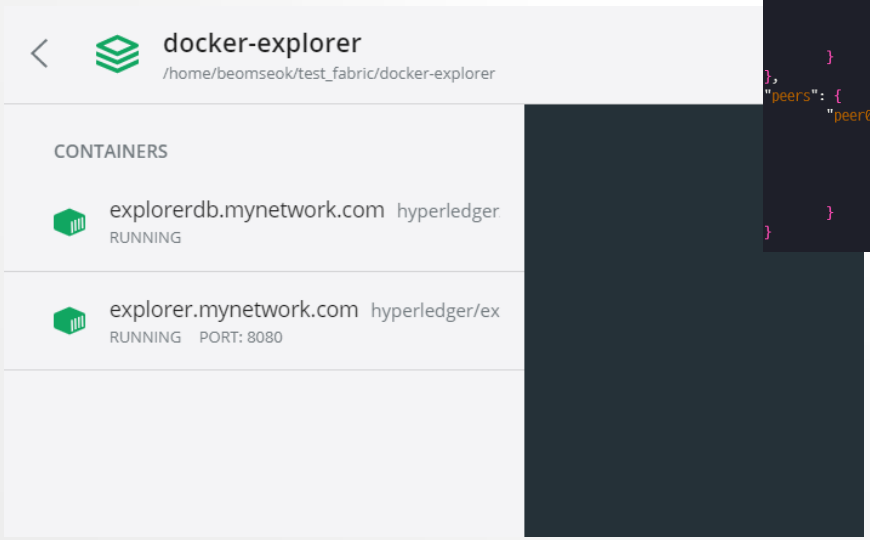
```
docker pull hyperledger/explorer
docker pull hyperledger/explorer-db

wget https://raw.githubusercontent.com/hyperledger/blockchain-explorer/master/examples/net1/config.json
wget https://raw.githubusercontent.com/hyperledger/blockchain-explorer/master/examples/net1/connection-profile/first-network.json -P connection-profile
wget https://raw.githubusercontent.com/hyperledger/blockchain-explorer/master/docker-compose.yaml

# 네트워크 실행 후
cp -r test_fabric/test-network/organizations/peerOrganizations docker-explorer/organizations/peerOrganizations
cp -r test_fabric/test-network/organizations/ordererOrganizations docker-explorer/organizations/ordererOrganizations

# 실행 - 8080 포트
docker-compose up -d
docker-compose down -v
```

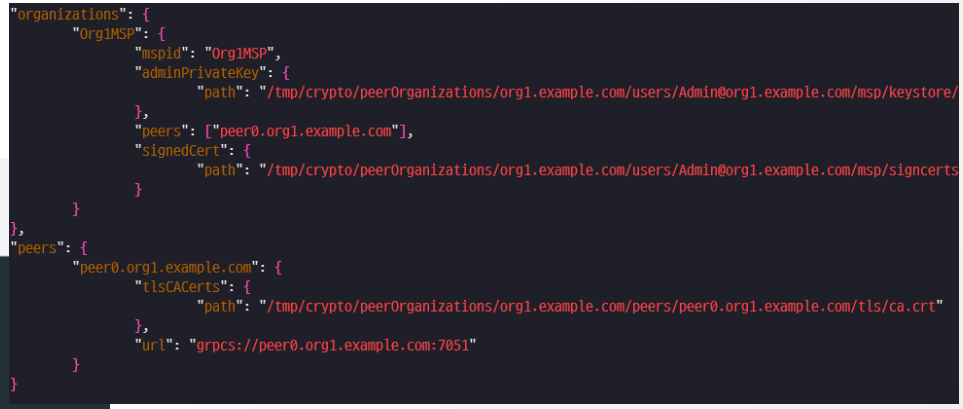
1. 블록체인 네트워크 환경 구성

2. Hyperledger Fabric, Hyperledger Explorer 연결

구성해야 하는 설정 파일

- config.json
- connection-profile/first-network.json
- docker-compose.yaml





1. 블록체인 네트워크 환경 구성

2. Hyperledger Explorer (블록체인 네트워크 담당자 모니터링 페이지) 실행화면

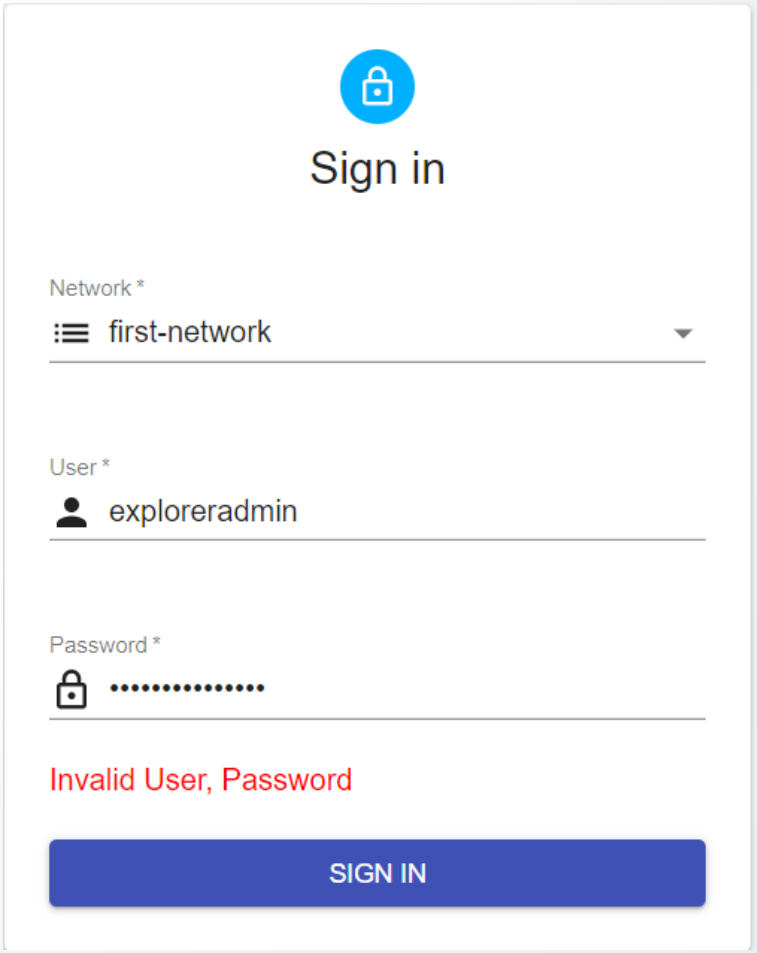
- 로그인

웹상에서 접속해 지정된 아이디와 비밀번호로

로그인할 수 있다. (관리자용 페이지이기

때문에 아무나 가입할 수 없고, CA 를 통해

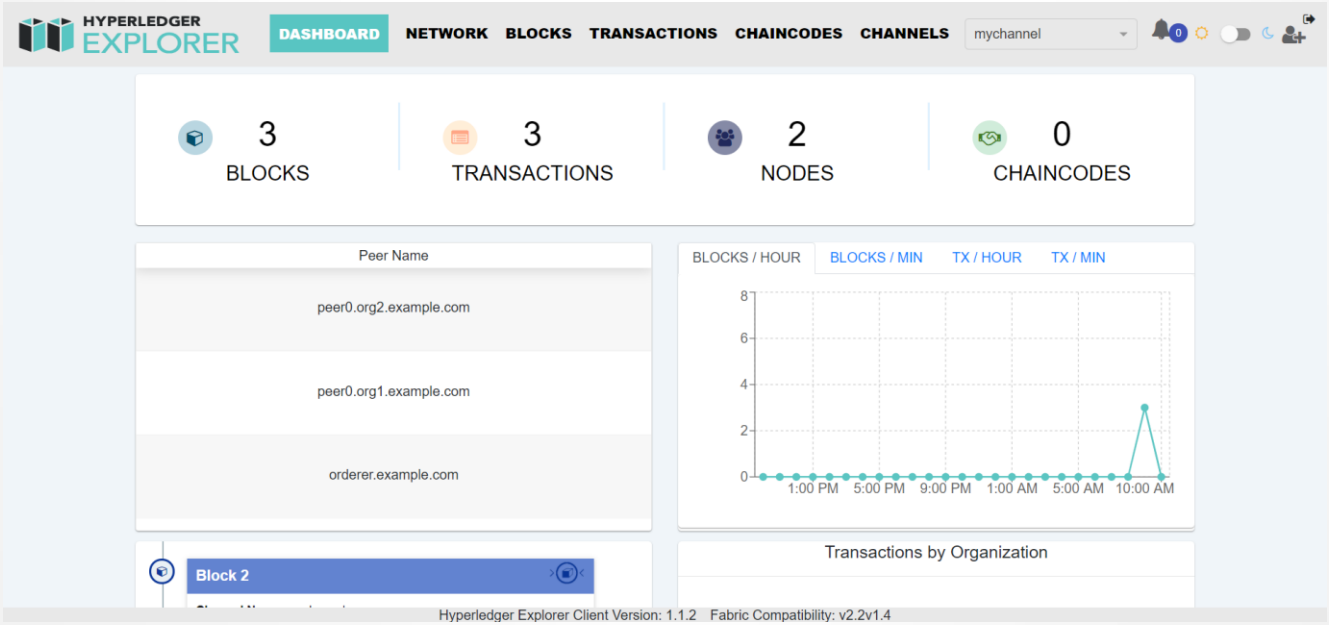
인증 받은 사람만 가입, 접속 가능)



1. 블록체인 네트워크 환경 구성

2. Hyperledger Explorer (블록체인 네트워크 담당자 모니터링 페이지) 실행화면

- 메인 페이지



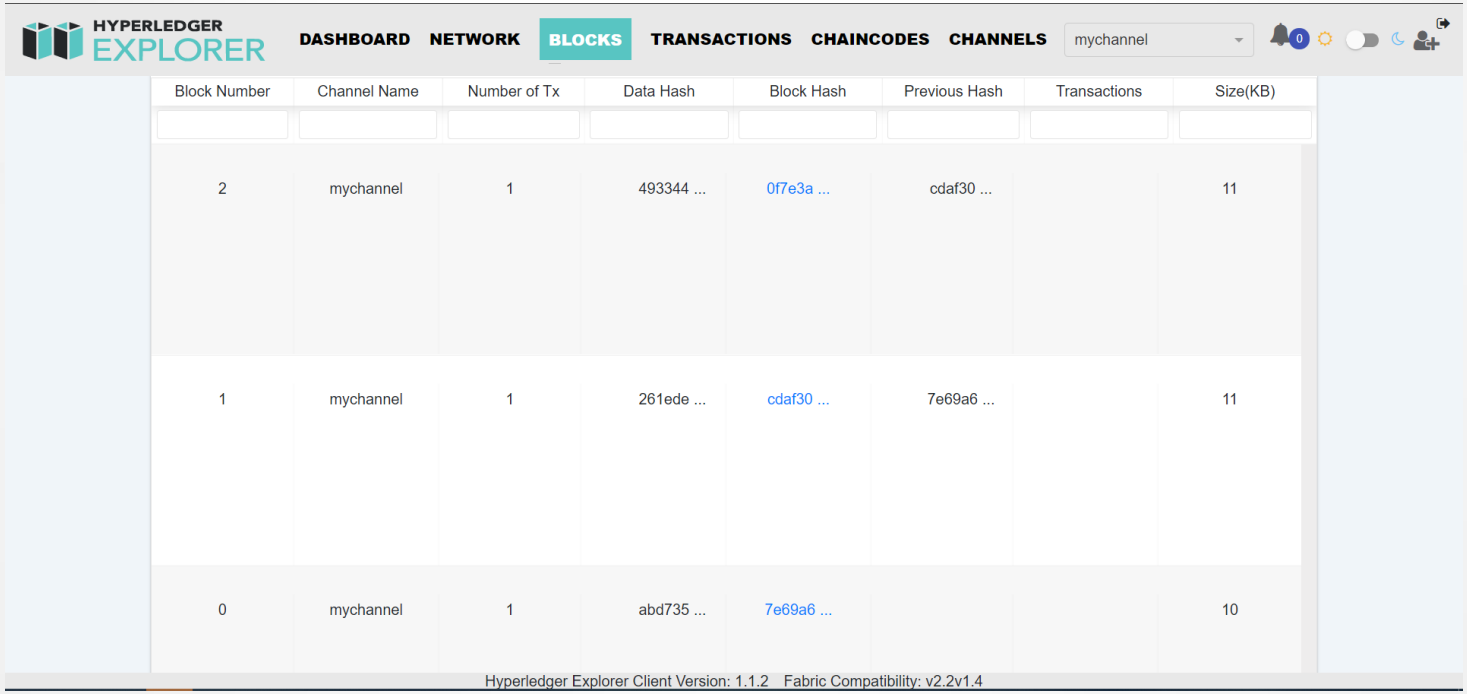
현재 운영중인 노드 개수 (Ordering Node 제외), 블록 개수, 트랜잭션, 채널에 설치된

체인코드 개수를 알 수 있고, 각 피어의 이름, 시간 별 블록 개수 변화, 트랜잭션 개수 변화를 그래프로 알 수 있다.

1. 블록체인 네트워크 환경 구성

2. Hyperledger Explorer (블록체인 네트워크 담당자 모니터링 페이지) 실행화면

- 블록 페이지

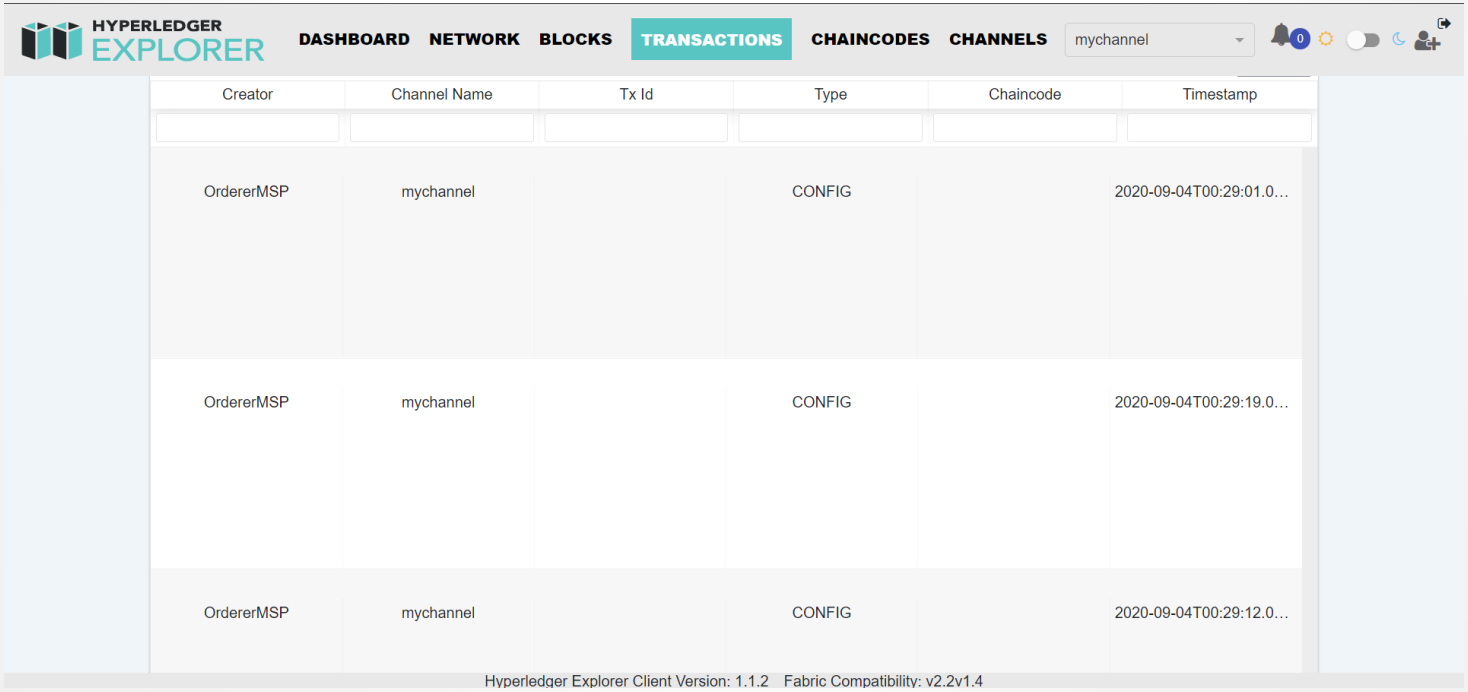


블록 생성 내역과 해시 값, 크기 등을 볼 수 있다.

1. 블록체인 네트워크 환경 구성

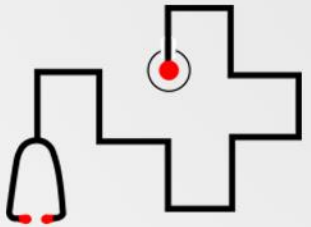
2. Hyperledger Explorer (블록체인 네트워크 담당자 모니터링 페이지) 실행화면

- 트랜잭션 페이지



트랜잭션 발생 내역과 Tx Id, Type 등을 조회할 수 있다.

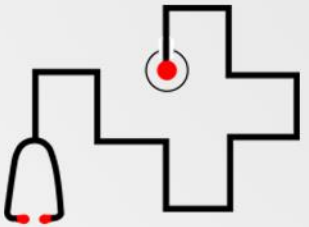
1. 블록체인 네트워크 환경 구성



3. Hyperledger Fabric 애플리케이션 작성 계획 (체인코드 실행을 통한 데이터 업로드, 조회 프로그램 개발 예정)

```
const fixtures = path.resolve(__dirname, '../../../../test-network');

async function main() {

    // Main try/catch block
    try {

        // A wallet stores a collection of identities
        const wallet = await Wallets.newFileSystemWallet( directory: '../identity/user/balaji/wallet');

        // Identity to credentials to be stored in the wallet
        const credPath = path.join(fixtures, '/organizations/peerOrganizations/org1.example.com/users/User1@org1.example.com');
        const certificate = fs.readFileSync(path.join(credPath, '/msp/signcerts/User1@org1.example.com-cert.pem')).toString();
        const privateKey = fs.readFileSync(path.join(credPath, '/msp/keystore/priv_sk')).toString();

        // Load credentials into wallet
        const identityLabel = 'balaji';

        const identity = {
            credentials: {
                certificate,
                privateKey
            },
            mspId: 'Org1MSP',
            type: 'X.509'
        }

        await wallet.put(identityLabel, identity);

    } catch (error) {
```

2. IPFS 네트워크 환경 구성



0. Prerequisites

OS: Linux in Window 10 (WSL2) 혹은 Linux

1. IPFS 설치 및 데몬 실행

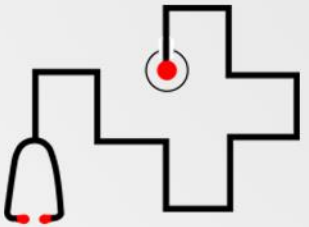
1. wget으로 설치용 tar 파일을 다운 (현재 Linux용 최신 버전은 `ipfs_v0.6.0`)

2. tar 파일을 unzip하고 생성된 폴더에서 설치용 bash 파일을 실행

3. IPFS 전용 repository를 init 해준다.

4. IPFS Daemon을 실행한다.

2. IPFS 네트워크 환경 구성

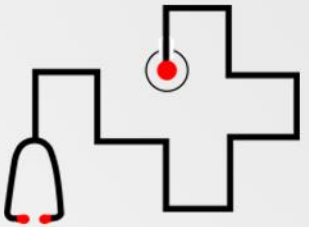


IPFS Daemon 실행에 성공한 화면.

자신의 노드와 연결된
다른 peer의 주소를
볼 수 있다.

```
/mnt/c/Users/win10 ipfs daemon
Initializing daemon...
go-ipfs version: 0.6.0
Repo version: 10
System version: amd64/linux
Golang version: go1.14.4
Swarm listening on /ip4/127.0.0.1/tcp/4001
Swarm listening on /ip4/127.0.0.1/udp/4001/quic
Swarm listening on /ip4/172.19.104.16/tcp/4001
Swarm listening on /ip4/172.19.104.16/udp/4001/quic
Swarm listening on /ip6/::1/tcp/4001
Swarm listening on /ip6/::1/udp/4001/quic
Swarm listening on /p2p-circuit
Swarm announcing /ip4/127.0.0.1/tcp/4001
Swarm announcing /ip4/127.0.0.1/udp/4001/quic
Swarm announcing /ip4/172.19.104.16/tcp/4001
Swarm announcing /ip4/172.19.104.16/udp/4001/quic
Swarm announcing /ip6/::1/tcp/4001
Swarm announcing /ip6/::1/udp/4001/quic
API server listening on /ip4/127.0.0.1/tcp/5001
WebUI: http://127.0.0.1:5001/webui
Gateway (readonly) server listening on /ip4/127.0.0.1/tcp/8080
Daemon is ready
```

2. IPFS 네트워크 환경 구성



2. js-ipfs API 를 통한 IPFS 사용

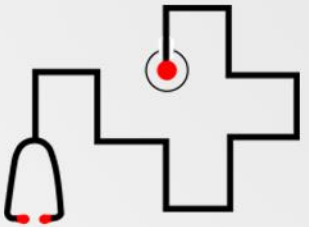
Node.js 프로그램에서 IPFS 네트워크를 사용하기 위해 js-ipfs-http-client API를 사용한다.

IPFS의 js-ipfs API에는 크게 js-ipfs와 js-ipfs-http-client 가 있는데, 더 가볍게 사용할 수 있는 js-ipfs-http-client API를 사용

IPFS에 이미지를 업로드하고, 업로드할 때 생성된 CID를 이용하여 다시 해당 이미지를 받을 수 있는 기능 구현.

- `js-ipfs` is a full implementation of IPFS, similar to `go-ipfs`. You can use it either as a command-line application or as a library to start an IPFS node directly in your program.
- `js-ipfs-http-client` is a smaller library that controls an IPFS node that is already running via its HTTP API. `js-ipfs` actually uses this library internally if it detects that another node is already running on your computer. You can also interact with the HTTP API directly using `fetch()` in a browser or a module like `request` in Node.js, but using this library can be much more convenient.

2. IPFS 네트워크 환경 구성



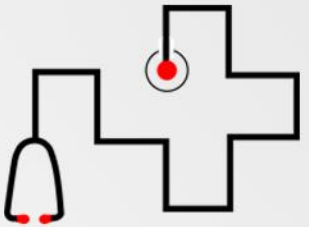
2. js-ipfs API 를 통한 IPFS 사용

imgAdder.js 에서는 이미지 파일의 경로를 읽어서 그대로 IPFS에 add 해준다.

add에 성공하면 IPFS상의 파일 경로(default는 CID)와 CID, 사이즈를 반환해준다.

```
PS C:\Users\win10\Desktop\graduPro\test_ipfs> node imgAdder.js
{
  path: 'QmeSgxdp8r98xeyJbbwqYrS8oZBuWZ1x1sRFKAn7Wgph97',
  cid: CID(QmeSgxdp8r98xeyJbbwqYrS8oZBuWZ1x1sRFKAn7Wgph97),
  size: 19770
}
```

2. IPFS 네트워크 환경 구성



2. js-ipfs API 를 통한 IPFS 사용

imgGetter.js 에서는 IPFS상에서 지정된 CID를 이용해 파일을 찾아 가져와준다.

get에 성공하면 IPFS 상의 파일 경로(default는 CID)와 파일의 버퍼를 반환해준다.

```
PS C:\Users\win10\Desktop\graduPro\test_ipfs> node imgGetter.js
{
  path: 'QmeSgxdp8r98xeyJbbwqYrS8oZBuWZ1x1sRFKAn7Wgph97',
  content: Object [AsyncGenerator] {}
}
```

이미지 파일을 제대로 요청했다면 반환받은 객체의 content를 적절히 가공하여 다시 이미지 파일로 만들 수 있다.

2. IPFS 네트워크 환경 구성

3. imgAdder.js와 imgGetter.js

```
addipfs('./images/e.png')
```

```
PS C:\Users\win10\Desktop\graduPro\test_ipfs> node imgAdder.js
{
  path: 'QmeSgxdp8r98xeyJbbwqYrS8oZBuWZ1x1sRFKAn7Wgph97',
  cid: CID(QmeSgxdp8r98xeyJbbwqYrS8oZBuWZ1x1sRFKAn7Wgph97),
  size: 19770
}
```

graduPro > test_ipfs > images

e

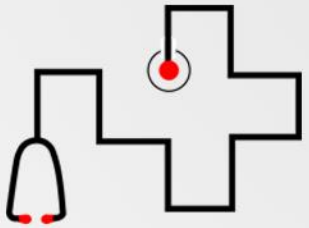
```
getipfs(cid, './return/ipfs_return.png')
```

```
PS C:\Users\win10\Desktop\graduPro\test_ipfs> node imgGetter.js
{
  path: 'QmeSgxdp8r98xeyJbbwqYrS8oZBuWZ1x1sRFKAn7Wgph97',
  content: Object [AsyncGenerator] {}
}
```

graduPro > test_ipfs > return

ipfs_return

3. 딥러닝 모델 구동



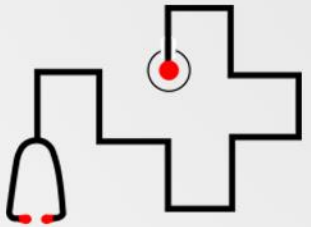
0. Prerequisites

- OS: Ubuntu 18.04
- Language: Python 3.6

1. 딥러닝 서버와 학습 클라이언트간 네트워크 구축

- 마이크로 서비스 스타일 아키텍처에서 다중언어 서비스를 효율적으로 연결하고, 효율적인 클라이언트 라이브러리 생성하는 gRPC 사용
- gRPC 에서는 클라이언트 응용프로그램을 로컬 객체인 것처럼 다른 컴퓨터의 서버 응용 프로그램에서 메소드를 직접 호출 할 수 있으므로 분산 응용 프로그램을 쉽게 만들 수 있음.

3. 딥러닝 모델 구동



2. ProtoBuf 코드 작성 및 skeleton 파일 생성

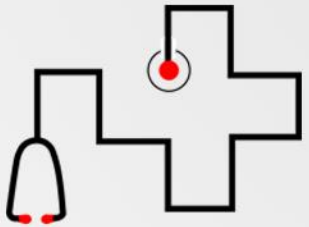
- 클라이언트와 서버 간 모델을 주고 받아야하기 때문에 바이트로 주고 받음.

- 클라이언트는 서버의 Updater 클래스에 있는 updateModel 메소드를 통해 모델을 주고 받음

- 학습의 종료, 재시작에 대한 메소드 추가 예정

```
syntax = "proto3";

package grpc;

service Updater {
  rpc updateModel (updateRequest) returns (updateReply) {}
}

message updateRequest {
  bytes model = 1;
}

message updateReply {
  bytes model = 1;
}
```

3. 딥러닝 모델 구동

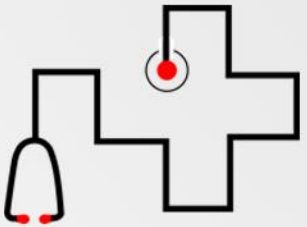


3. 클라이언트

1. 학습 모델은 Unet을 사용.
2. 서버에 연결
3. 학습
   - Optimizer: Adam
   - Learning rate: 0.001
   - Loss: cross entropy
4. Evaluate: 이전 evaluate의 loss와 비교
5. 발전되었으면 학습한 모델, 아니면 이전의 모델을 바이트로 변환 후 서버에 전송
6. 서버에서 response를 모델을 받아서 현재 모델에 적용
7. 서버에서 학습 명령이 있으면 학습, 아니면 학습 명령을 대기

```
learning...
/home/yang/anaconda3/envs/py36/lib/python3.6/site-packages/sklearn/ut
  warnings.warn(msg, category=FutureWarning)
72/72 [==============================] - 210s 3s/step - loss: 0.4205
evaluating...
58/58 [==============================] - 4s 67ms/step - loss: 0.6804
loss improved
sending Model...
Received Model...
learning...
```

3. 딥러닝 모델 구동



3. 서버

1. 서버 실행
2. Client에서 model request를 받음
3. Server의 메인 model과 request 받은 모델을 합침
4. 합친 결과를 client로 response