프라이빗 네트워크 구축

Tags Blockchain

채굴 원리

비트코인은 은행, 신용카드 회사 등의 금융기관 중개가 없이 안전한 거래가 이뤄지도록 모든 거래내역을 장부에 기록하여 모든 사용자들이 공유할 수 있는 방식을 선택했다. 비트코인 시스템은 전체 거래내용을 10분 단위로 모아 장부에 기록하는데 거래내용은 암호화되어 있기 때문에 누군가 암호를 풀어 장부를 기록해야 한다. 암호는 수많은 계산과 검토가 필요한 어렵고 번거로운 일이기 때문에 암호를 풀어 장부에 기록하는 권리와 그 대가인 신규 발행된 비트코인을 한 사람에게 주는데 이 과정은 마치 광부가 광산에서 곡괭이질을 거듭한 끝에 금을 캐내는 것과 비슷하다고 하여 채굴이라고 한다.

'작업'이란 '채굴'에 이르기까지 연산 과정을 뜻한다. 채굴자들은 컴퓨터로 복잡한 수식을 풀어 조건에 맞는 해시값을 찾는 과정을 반복한다. 이 경우 모든 노드들이 찾아낸 해시값을 검증하고 승인하는 과정을 거쳐 블록에 거래 내역을 저장한다. 따라서 모든 노드들의 승인을 거쳐야 하기 때문에 거래 내역을 속이기가 힘들다는 장점이 있다. 이런 점에서 작업증명 합의 알고리즘은 블록체인이 가지는 탈중앙화라는 본질을 가장 잘 살린 합의 방식이다. 그러나 이런 과정 때문에 거래 처리 속도가 늦어진다는 한계가 있다. 또한 채굴에 필요한 에너지 소비가 심하다는 것도 단점이다. 이 때문에 일정 조건에 따라 블록 생성에 참여하는 노드들을 제한하는 지분증명방식이 등장했다

프라이빗 네트워크 구축

우리는 블록체인을 완성하기 위해 네트워크를 구축해야 한다.
51%의 지분이 있는 쪽이 진짜임을 믿는 시스템이기 때문에 public network라면 노드들이 많으면 많을수록 좋다. 그 만큼 해커가 해킹을 시도하는 것이 힘들어지기 때문이다.

먼저 networkid를 정해야 한다. 어떤 네트워크 아이디는 사용되고 있고, 우리는 private으로 구축하기 위해 사용하지 않고 있는 네트워크 id를 택한다. 여기에서는 921로 선택했다.

작업 증명을 위한 합의 알고리즘을 골라야 한다. PoW와 PoA가 있다.

작업 증명 합의 알고리즘은 더 공부해서 정리

genesis block을 만들어야 한다.
모든 블록체인은 genesis bloc에서 부터 시작한다. 만약 default 세팅으로 geth를 통해 run을 하게 되면 Mainnet genesis을 사용하여 메인넷에 붙게 된다. private network를 위해서는 다른 genesis block을 사용해야 한다. genesis block은 genesis.json 파일로 geth를 통해 구성된다.

geth init 을 통해 초기화를 한번 해준다음에 네트워크를 세팅해준다.

노드 구축에 대해서는 여러 방법이 있다. bootnode를 만들고 이것에 다른 노드들을 연결하여 bootnode가 broadcast하는 방법이 있고, 혹은 각각의 노드들을 따로 구성한 다음에 addPeer을 통해 연결하는 방법이 있다. 나는 후자의 방법을 사용했다.

각각의 노드에 geth를 실행시켜서 네트워크를 구성하고 geth attach를 통해 터미널을 들어가서 첫번째 노드에서 admin.nodeInfo.enode를 하고 이 값으로 다른 노드들에서 admin.addPeer("enode값")으로 연결해준다.

Geth 설치

우분투의 경우 다음과 같이 실행한다.

```
sudo add-apt-repository -y ppa:ethereum/ethereum
sudo apt-get update
sudo apt-get install ethereum
```

genesis.json 파일을 다음과 같이 작성한다.
coinbase란 채굴을 통해 얻는 이득을 가져가는 계정을 말하고,
nonce는 중복 시도를 막기 위함이며,
difficulty는 채굴을 하는 난이도 이다. 이걸 높게 잡으면 하나의 트랜잭션이 엄청 오래걸리는 것을 경험할 수 있다.
chainId는 네트워크 id이다. 메인넷은 1이며, 그 외의 여러 테스트넷고 네트워크 id가 있다. 하지만 만약 private network를 구성하고 싶다면 chainId를 아무도 사용하지 않는 넘버로 정하면 된다.
alloc에는 계정 정보가 들어간다. 만약 alloc에 아무정보를 넣지 않으면 geth를 실행하고 콘솔창에서 계정을 만들어도 되며, 만들어진 계정은 data/keystore 폴더에 json형태의 키값이 저장된다.
이 키값을 그냥 들여보면 private key를 알 수 없다. 우리는 계정을 만들 때 우리만의 비밀번호를 입력하는데 이 비밀번호를 알고 key파일을 가지고 있어야 비로소 private key를 알아낼 수 있다.

```
{
        "config": {
                "chainId": 921,
                "homesteadBlock": 0,
                "eip150Block": 0,
                "eip155Block": 0,
                "eip158Block": 0
        },
        "nonce": "0xdeadbeefdeadbeef",
        "gasLimit": "9999999",
        "difficulty": "0x0",
        "coinbase": "0x0000000000000000000000000000000000000000",
        "alloc": {
                "0xD8Ff804aB2Be363F046e7D19064d60626Cf004d3":{
                        "balance":"1000000000000000000"
```

```
            }
        }
}
```

주의할 점은 위의 genesis.json이 티끌하나도 다르면 서로 연결이 안된다!!!!! 주의할것!!!

```
geth --datadir data/ init genesis.json

geth attach http://localhost:8545 // 그 후에 attach 를 통해서 콘솔로 접근

ps ax | grep geth // 백그라운드 geth 노드가 켜져있는지 확인하기

pkill geth // geth 노드 끄기
```

명령어를 통해 우리가 만든 설정파일인 genesis.json을 기준으로 geth의 설정이 세팅이 된다. 만들어진 세팅은 data/geth라는 폴더 안에 다 들어가 있다. 이제 geth에서 일어나거나 명령하는 것들이 이 폴더에서 세팅이 될 것이다.
이제 준비가 거의 완료되었다. 실제로 네트워크 노드를 구성해보자.

기준이 될 노드는 다음과 같이 nohup을 사용하여 데몬 프로세스로 돌린다.

```
nohup geth --networkid 921 --datadir ./data --allow-insecure-unlock --port 30303 --http --http.port 8545 --http.addr 0.0.0.0 --http.vhosts "*" --http.corsdomain "*" --h
```

그리고 다른 이을 노드들은 다음과 세팅한다.

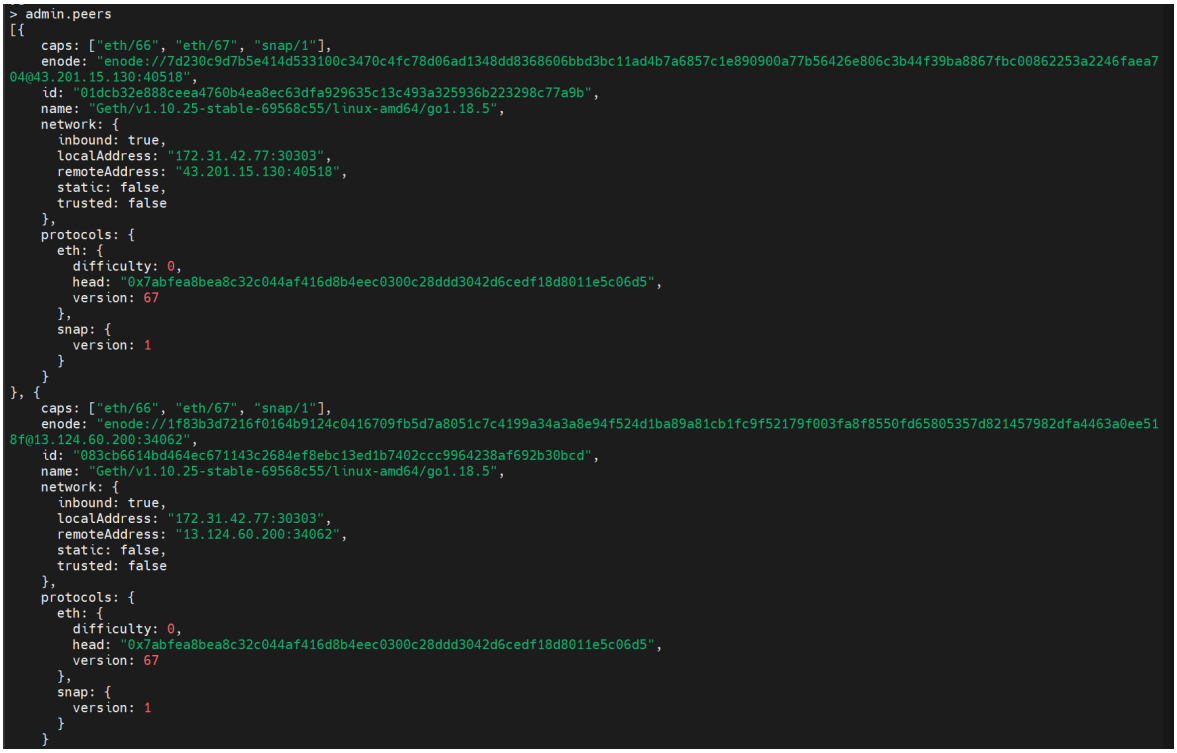
```
nohup geth --networkid 921 --datadir ./data --allow-insecure-unlock --port 30303 --http --http.port 8545 --http.addr 0.0.0.0 --http.corsdomain "*" --http.api "admin,net
```

사실 별 차이 없다. —nat을 사용해서 외부 ip를 노출시켰느냐 아니냐의 차이이다. 위것으로 해줘도 사실 상관은 없다.

이제 콘솔안에서 채굴을 할 계정과 채굴의 보상을 얻을 계정을 설정해준다.

- admin.nodeInfo : 이 노드에 대한 정보
- admin.addPeer() : 다른 노드들을 기준노드에 연결을 해야 한다. 그것을 하기 위해 addPeer에 기준노드의 admin.nodeInfo.enode 값을 넣어줘서 서로 연결한다.

```
admin.addPeer("enode://dc96eb1d8ef0c46b4a36e0fa98c71ba190b19b992a4139355c2e4875eb2b3253dc2c9d435fadd7bd141289f495bb45fb3fc76e163dc9de9fc7d06cc90f03071a@43.201.23.2
```



- net.peerCount를 통해서 서로 연결되었는지 확인한다.
- eth.accounts : 이 노드의 계정들
- eth.coinbase : 이 노드의 coinbase
- miner.setEtherbase(eth.accounts[1]) : 계정들중 2번째 계정을 이더 베이스로 설정
- eth.getBalance(eth.accounts[0]) : 이더 잔고 확인
- eth.blockNumber : 블록체인의 블록 수 확인. 블록 수는 채굴을 할 수록 늘어난다.

- miner.start("thread_num") : 채굴할 때 사용할 thread 수를 설정하고 채굴을 한다.
- eth.mining : 이 노드가 채굴을 하고 있는 지 여부 확인
- eth.hashrate : 해시 속도 확인, 해시 속도는 채굴하는 데 사용하는 연산력을 나타내는 값으로 단위는 hash/s(초) 이다.
- admin.peers : 이 노드의 피어들의 정보를 출력한다.
- eth.getTransaction("transaction hash") : 트랜잭션에 대한 정보를 출력
- personal.newAccount() : 새로운 계정 생성
- miner.setGasPrice() : 채굴할때 받는 이더 값 세팅

참고로 geth 노드 구성을 genesis.json을 사용하여 새롭게 구축한 경우엔 sudo reboot를 통해 재부팅을 한번 해줘야 한다.

참고자료

https://lbmmbl.tistory.com/10

https://dejavuqa.tistory.com/234

https://ethereum.stackexchange.com/questions/366/how-can-i-run-go-ethereum-as-daemon-process-on-ubuntu

https://blockgeeks.com/two-node-setup-of-a-private-ethereum/#Step_1_Launch_two_EC2_instances

https://dejavuqa.tistory.com/262

https://github.com/ethereum/wiki/wiki/[Korean]-White-Paper

https://geth.ethereum.org/docs/install-and-build/installing-geth

https://lbmmbl.tistory.com/5

solidity로 스마트 컨트랙트 코드 (.sol) 짜기

.sol 파일을 자바에서 사용할 수 있도록 Wrapper (.java) 파일로 바꾸기

먼저 solc 를 설치하고, web3j도 설치해준다음에 아래와 같은 명령어를 만든다.

smart-contract 폴더로 가서 npm i를 통해 필요한 모듈들을 설치해주고 그 후에 보면 node_modules가 있는 것을 확인할 수 있다.
이제 solcjs를 사용하여 .sol 파일들을 abi, bin으로 바꾼다음에 다시 .java 파일로 바꿔준다.

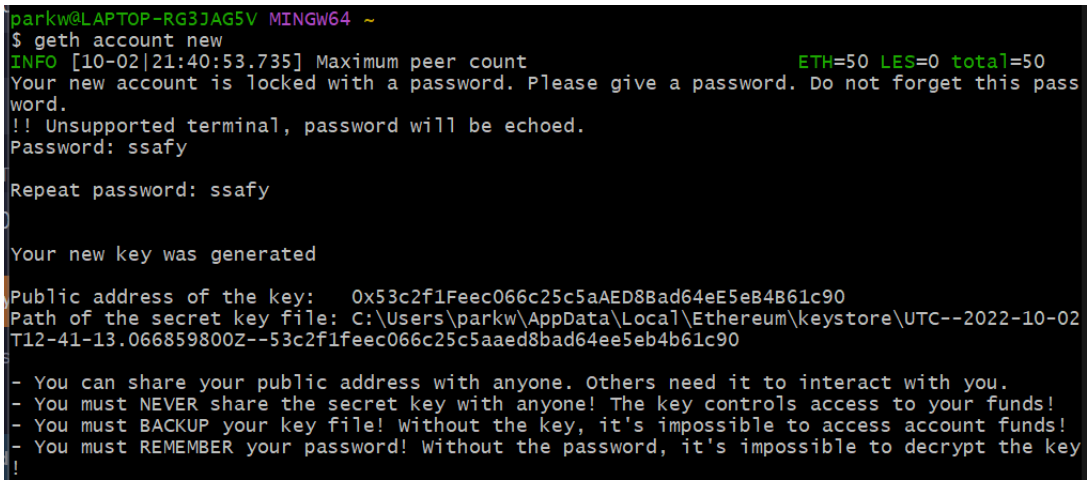
```
solcjs --bin --abi --include-path node_modules/ --base-path . -o ./contracts contracts/SingleBungle.sol
```

```
web3j generate solidity -a contracts_IERC20_sol_IERC20.abi -b contracts_IERC20_sol_IERC20.bin -o java -p com.sgbg.blockchain.wrapper
```

web3j와 wrapper 클래스를 사용하여 서비스 완성

프라이빗 네트워크와 모든 파일들 연결

먼저 스마트 컨트랙트 중 Cash.sol은 한 번만 배포가 되어야 하며 그 한 번도 admin 계정이 있어야 한다. 일반 유저가 코인을 충전하는 것은 우리 서비스에서 코인을 그냥 주는 것 같지만 사실은 admin 지갑이 있고 이 지갑에 많은 돈이 들어가 있다. 그리고 일반 유저가 충전을 신청하면 admin 지갑의 코인을 일반 유저에게 주는 식이다. 따라서 가장 먼저 admin 지갑을 geth를 사용하여 생성한다.



위와 같이 터미널에서 geth 명령어를 사용하여 계정을 새롭게 만들면 시크릿 키 파일이 있는 곳을 알려준다. 이제 이 파일을 admin.wallet으로 이름을 바꾸고 프로젝트의 resources에 넣어준다.
이제 application.properties에서 참조를 할 수 있게 되었다.

```
eth.admin.address=53c2f1feec066c25c5aaed8bad64ee5eb4b61c90
eth.admin.wallet.filename=admin.wallet
```

이렇게 설정을 해놓고, 이제 Cash.sol을 배포하러 가보자.

근데 geth를 통해서 계정을 만들어도 가나슈에서는 account로 등록되지 않는다.
따라서 그냥 가나슈 콘솔에 있는 계정 중 처음 계정을 사용하여 admin으로 만들었다.

이 이유는 위에서 만든 geth address와 지금 그냥 테스트로 진행하는 가나슈와는 다르기 때문이다. 위의 geth는 메인넷에서 사용하는 계정이 아닐까..

매번 네트워크를 실행할 때마다 admin을 바꿔서 해야 한다.

먼저 프라이빗 네트워크를 구동하면 지갑인 계정을 만든다.

wallet.txt 를 만들어서 저장을 해놓고 application.properties를 사용해서 하고
admin 계정의

리믹스에 들어가서 먼저 지갑을 만들어서 walletFile을 만들어서 앱에 넣어놓고
.sol을 가지고 가서 private network에 연결하여 배포를 먼저 하고

궁금한점

- singlebungle.sol을 보면 최소 모집 금액이 있는데 왜 이더가 단위인지??
- 프라이빗 네트워크를 어떻게 구축할 것인지
- 트랜잭션 id

프로젝트 애플리케이션 파일 (1)

가나슈 테스트넷 도커이미지를 사용하여 배포

```
docker run -d -p 8545:8545 --ip=172.26.10.156 --name ganache trufflesuite/ganache-cli:latest --gasPrice 0x0
```

account : 0xa702dEB6647d077988dc1353C323df10BE0BaA57

Private Keys : 0xc22bc8d3d4b9a9c375729602097c90bc54646839b7f2643ab48c7645871635c4

/jenkins/workspace 에 있는 application.properties 에서 account와 private key값을 바꾼다.

jenkins 빌드를 다시 한번 눌러준다.

GAS Limit : 본 송금 '작업'에서 소비되는 가스량
(estimated 한 수치라서 변경가능합니다. 하지만 넘 작게하면 거부됨)

GAS Price : 내가 가스당 지불할 가격
(경매처럼 내가 금액을 제안하는 것임)

MAX TOTAL : GAS Limit * GAS Price 로 형성된 최종지불금액