

00.블록체인기간일정

노트북: 블록체인

만든 날짜: 2019-10-29 오후 2:07

업데이트: 2019-10-29 오후 2:10

작성자: 비니문

블록체인기간일정표

시간	내용	비고
09:00 ~ 09:10	조회/프로젝트 컨설턴트 리뷰	
09:10 ~ 09:20	스프린트 계획	
09:20 ~ 09:50	프로젝트 진행	
10:00 ~ 11:20	프로젝트 집중 시간 1부	이동금지
11:40 ~ 12:15	프로젝트 진행	취업 컨설팅 가능
12:15 ~ 13:15	점심시간	
13:15 ~ 13:50	프로젝트 진행	취업 컨설팅 가능
14:00 ~ 14:50	프로젝트 진행	취업 컨설팅 가능
15:00 ~ 16:40	프로젝트 집중 시간 2부	이동금지
17:00 ~ 17:30	프로젝트 진행	
17:30 ~ 18:00	일일 개발 내용 정리/코드리뷰/종래	

01.블록체인 주요 용어

노트북:	블록체인	업데이트:	2019-10-29 오후 2:10
만든 날짜:	2019-08-19 오전 11:19		
작성자:	비니문		

블록체인에 들어가기 전에 알아두면 좋은 주요 용어들

암호화폐 (Cryptocurrency, 가상화폐)

- 비트코인 및 비트코인 기술 기반의 전자화폐를 구현할 때 디지털 서명, 해시체인 등 암호학 기반 기술을 많이 사용함. 블록체인 기반의 전자화폐도 암호화 기술을 사용하므로 이들을 통칭해 암호화폐라고 함

명목화폐 (Fiat Money)

- 일반적인 화폐. 파이트 화폐라고도 함

비트코인 (Bitcoin)

- 전자화폐 시스템으로서 기존 전자화폐와는 달리 관리자가 없이도 자율적으로 동작하는 분산 시스템

이더리움 (Ethereum)

- 블록체인 기반 애플리케이션을 개발하고 운용하는 플랫폼
- 튜링 완전한 프로그래밍 언어 기반의 애플리케이션을 개발할 수 있음
- 비트코인과는 근본적으로 구조가 다름

블록체인 (BlockChain)

- 관리자 없이 자율적으로 동작하는 분산 시스템 기술을 통칭
- 화폐 거래 내역을 '블록'이라는 데이터 단위로 저장한 후 해당 블록의 해시값을 다른 블록에 저장시켜 체인 형태의 연결고리를 만들

블록체인 애플리케이션 (Blockchain Application)

- 스마트 계약 기반 암호화폐 및 디지털 애셋의 거래, DApps 개념을 포함

DApps

- 탈중앙화 애플리케이션의 줄임말
- 특정 관리자가 없이도 계속 동작하면서 스마트 계약을 실행하는 애플리케이션

스마트 계약 (Smart Contract)

- IT 기술을 이용해 계약 내역을 자동으로 실행하는 것
- 예) 전자화폐의 잔액이 일정 액수 이하면 신용카드로 자동으로 충전하는 서비스도 스마트 계약의 하나

주소 (Address)

- 공개 키 암호로 암호화폐를 받거나 보내는 단위로 다루는 임의 문자열
- 공개 키를 바탕으로 만들고 비밀 키가 있는 프로그램으로 암호화폐의 이용 권한을 얻을 수 있음

지갑 (Wallet)

- 암호화폐 거래에 필요한 개인 키를 저장한 공간을 뜻함
- 암호화폐 자체는 블록체인에 공유해 저장

디지털 자산 (Digital Asset)

- 다이아몬드와 고급 자동차 등의 실제 자산 소유권 거래 내역을 블록체인으로 디지털화한 것을 말함

거래 (Transaction)

- 일반적으로 시스템 안에서 더 나눌 수 없는 처리 단위를 의미
- 블록체인에서는 코인과 토큰 소유권을 포함하는 데이터를 주고받는 것을 뜻함
- 작성자의 전자 서명을 적용해 코인 및 토큰 발행을 증명하거나, 내용을 조작한 사실이 없음을 보장하는데 사용

블록 (Block)

- 여러 거래를 모아 만든 데이터 단위
- 거래를 블록에 저장하면 올바른 거래인지 검증하며, 뒤에서 설명할 작업 증명 알고리즘 등을 이용해 이중 지급을 막음

블록 높이 (Block Height)

- 새로운 블록을 생성할 때는 먼저 생성한 블록의 해시값 (이전 거래 기록 등)을 저장해야 함
- 다른 블록의 해시값을 포함해 연결된 블록의 전체 개수를 의미함
- 블록 높이가 0이면 맨 처음 생성한 블록이며 이를 제네시스 (Genesis)블록이라고 함
- 거래와 블록 안 타임스탬프는 블록을 만든 사람이 자유롭게 설정가능 하지만 타임스탬프 기록으로 결정된 블록 생성 순서는 신뢰할 수 없음
- 기존 블록의 해시값을 새로 생성한 블록에 저장해 블록 생성 순서의 신뢰도를 보장

타임스탬프

- 날짜와 시간을 나타내는 문자열
- 블록체인은 실제 거래를 생성한 시각과 블록에 정식으로 저장한 시각을 타임스탬프에 저장하는데 두 기록이 일치하지 않아 문제가 발생하기 때문에 주의해서 다뤄야 함

해시값 (Hash Value)

- 암호화폐를 구분하는 작은 크기의 데이터
- 원본 데이터에서 해시값을 계산하는 함수를 해시 함수
- 암호화폐의 블록 생성이나 주소 계산 등에서 암호화 해시 함수로 만든 해시값을 많이 활용함

채굴 (Mining)

- 비트코인과 블록체인 기반 암호화폐는 누구나 새로운 블록 생성에 참여해 보상으로 암호화폐를 얻고 그러한 행동을 채굴이라고 함
- 채굴자는 Miner이라고 함

작업 증명 (Proof of Work, PoW) 알고리즘

- 채굴했다는 사실을 증명한 후에 블록을 생성하게 만드는 방법으로서 비트코인에서 채굴할 때 사용하는 알고리즘
- 동시에 증명 없이는 블록을 생성할 수 없게 하는 역할

- 그 이외에 지분 증명(Proof of Stake, PoS)와 존재 증명(Proof of Existence, PoE)알고리즘이 있음

메인넷(Main Net)과 테스트넷(Test Net)

- 메인넷
 - 독립적인 암호화폐로 인정하는 프로그램을 출시·운용하는 네트워크
 - 수수료를 내야 하거나 배포한 프로그램을 제거할 수 없는 문제가 발생
- 테스트넷
 - 블록체인 애플리케이션을 개발할 때 사용하는 메인넷과 같은 구조의 네트워크
 - 작업 증명 알고리즘을 적용한 롱튼(Ropsten)
 - 권한 증명 알고리즘을 적용한 코반(Kovan), 링크비(Rinkeby)

코인(Coin)과 토큰(Token)

- 코인
 - 메인넷이 있는 블록체인 시스템에서 발행한 암호화폐
- 토큰
 - 메인넷의 블록체인 시스템을 빌려 독자적인 암호화폐
- 보통 토큰 사용이 활발해지면 별도의 메인넷을 만들어 코인으로 승격시켜줌

합의(Consensus) 알고리즘

- 합의
 - 분산 시스템의 모든 프로세스가 같은 결과값을 결정하는 과정
 - 시스템에서 발생하는 에러를 막고 무결성을 보장
- 블록체인 시스템은 누구나 블록을 생성할 수 있는 구조이므로 블록의 소유권과 생성 순서를 결정하는 합의 알고리즘을 사용
- 블록체인은 블록 높이가 클수록 확장한 거래 내역이 바뀔 확률이 낮아지는 '확률적 합의' 개념의 알고리즘을 도입해 관리자가 없는 분산 시스템을 구현

확정(Confirmation)

- 거래를 블록에 저장시켜 기존의 거래 내역과 새로운 거래 내역을 검증하는 것
- 거래 내역 검증 작업은 어떤 거래 하나가 저장된 블록과 연결된 다른 블록의 개수만큼 반복
 - 연결된 블록의 개수를 확정 횟수라고도 함
- 블록체인은 확률적 합의를 적용하므로 확정 횟수가 많을수록 확정된 거래 내역의 신뢰도를 보장할 수 있음
 - 확정 횟수를 신뢰도를 나타내는 지표로도 이용

가스(Gas)

- 이더리움에서 애플리케이션을 실행할 때 지급하는 네트워크 수수료
- 이더리움을 유지하는 비용으로도 이용

ERC20

- Ethereum Request for Comments의 줄임말
- 이더리움 네트워크의 개선안을 제안하는 EIPs에서 관리하는 공식 프로토콜

02.블록체인 프로젝트_환경구성 가이드1

노트북: 블록체인
만든 날짜: 2019-08-19 오후 4:23
작성자: 비니문
URL: <https://ysc1230.tistory.com/12>

업데이트: 2019-09-12 오후 3:55

VSCode Terminal

PS c:\>로 바로 나가기

```
cd ../../
```

새로운 로컬 디렉토리 생성

```
mkdir ./dev/eth_prac001
```

디렉터리: c:\dev

Mode	LastWriteTime	Length	Name
d----	2019-08-19 오후 4:23		eth_prac001

파일 생성 확인하고 생성한 파일로 들어가기

```
ls
```

디렉터리: c:\

Mode	LastwriteTime	Length	Name
d----	2019-07-23 오전 11:15		bini
d----	2019-08-19 오후 4:23		dev
d----	2019-07-08 오전 10:13		errorwebmobliesecond
d----	2019-07-15 오후 5:56		googlekey
d----	2019-06-25 오후 12:00		HashiCorp
d----	2019-05-23 오후 9:15		Intel
d----	2018-09-15 오후 4:33		PerfLogs
d-r--	2019-08-16 오후 7:19		Program Files
d-r--	2019-08-16 오후 3:43		Program Files (x86)
d----	2019-08-19 오후 1:51		project_blockchain
d----	2019-07-16 오전 9:35		study
d----	2019-05-23 오후 8:46		Temp
d----	2019-07-26 오후 4:46		test
d-r--	2019-07-24 오후 5:30		Users
d----	2019-08-07 오후 7:33		UsersmulticampusAppDat
d----	2019-08-14 오후 2:58		vue_study
d----	2019-08-14 오후 6:02		Windows
d----	2019-07-05 오후 2:58		블록체인
d----	2019-08-19 오후 1:30		새 폴더
-a----	2019-07-16 오후 5:45	184940971	atomfile.zip
-a----	2019-08-05 오후 5:56	157202356	final_webmobile.7z
-a----	2019-05-23 오후 9:12	3033	rhdssetup.log
-a----	2019-05-23 오후 9:15	189	Setup.log
-a----	2019-08-14 오후 3:32	21590669	vue_study.zip

생성된 파일로 들어와서 vagrant init

그리고 Vagrantfile 수정

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

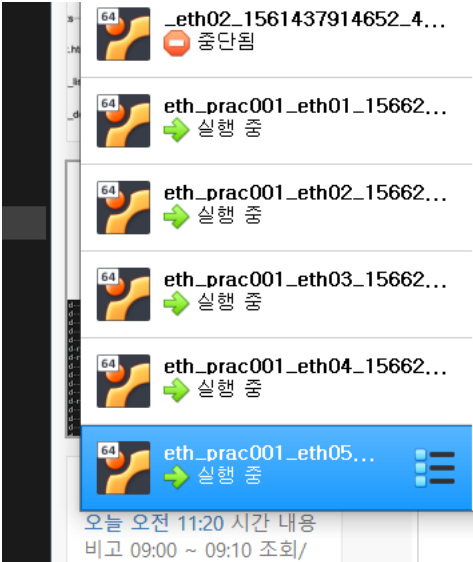
# All Vagrant configuration is done below. The "2" in Vagrant.configure
# configures the configuration version (we support older styles for
# backwards compatibility). Please don't change it unless you know what
# you're doing.

Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/bionic64"
  vms.map do |key, value|
    name = key.to_s
    ip_num, mem = value
    config.vm.define "#{name}" do |node|
      node.vm.network "private_network", ip: "192.168.50.#{ip_num}"
      node.vm.hostname = "#{name}"
      node.vm.provider "virtualbox" do |nodev|
        nodev.memory = "#{mem}"
      end
    end
  end
end
```

그리고 구동

vagrant up

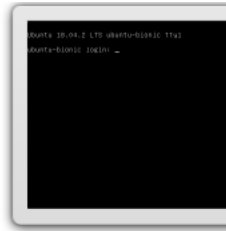
그리고 virtualBox실행 시켜서 5개의 머신 확인하기



이름: eth_prac001_eth05_15662
02612502_10645
운영 체제: Ubuntu (64-bit)
설정 파일 위치: C:\Users\Wmulticampus\VirtualBox VMs\Weth_prac001_eth05_1566202612502_10645

시스템
기본 메모리: 2048 MB
프로세서: 2
부팅 순서: 플로피 디스크, 광 디스크, 하드 디스크
가속: VT-X/AMD-V, 네스티드 페이징, PAE/NX, KVM 반가상화

디스플레이
비디오 메모리: 16 MB
그래픽 컨트롤러: VBoxVGA
원격 데스크톱 서버: 사용 안함



별도의 terminal에서 가상머신 접속 및 이더리움 설치
디렉토리로 들어가기

가상머신 접속하기

```
vagrant ssh eth01
```

가상머신에 `Geth` 설치

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

설치 후 설치 확인

```
geth version
```

```
vagrant@eth01:~$ geth version  
INFO [08-20|00:42:20.584] Bumping default cache on mainnet  
WARN [08-20|00:42:20.584] Sanitizing cache to Go's GC limits  
Geth  
Version: 1.9.2-stable  
Git Commit: e76047e9f5499b58064bddde514dd3119a090adf  
Architecture: amd64  
Protocol Versions: [63]  
Network Id: 1  
Go Version: go1.11.5  
Operating System: linux  
GOPATH=  
GOROOT=/usr/lib/go-1.11  
vagrant@eth01:~$
```

가상머신 내 디렉토리 생성 및 GENESIS 블록파일 생성

```
mkdir -p dev/eth_localdata  
cd dev/eth_localdata  
vi CustomGenesis.json
```

point: vi 명령어 공부하기

a 키 누르고 붙여넣고 esc : wq하기

```
{  
  "config": {  
    "chainId": 15150,  
    "homesteadBlock": 0,  
    "eip155Block": 0,  
    "eip158Block": 0  
  },  
  "difficulty": "0x10",  
  "coinbase": "0x000000000000000000000000000000000000000000000000",  
  "gasLimit": "9999999",  
  "alloc": {},  
  "extraData": "",  
  "nonce": "0xdeadbeefdeadbeef",  
  "mixhash": "0x000000000000000000000000000000000000000000000000",  
  "parentHash": "0x000000000000000000000000000000000000000000000000",  
  "timestamp": "0x00"  
}
```

```
geth --datadir /home/vagrant/dev/eth_localdata/ init /home/vagrant/dev/eth_localdata/CustomGenesis.json
```

파일들 생성됐는지 확인

```
tree
```

트리설치(새로운 가상머신이기에 때문에)

```
apt install tree
```

관리자로 설치해야 할림

```
sudo apt install tree
```

Go Ethereum 구동

원래는

```
geth --networkid 15150 --maxpeers 5 --datadir /home/vagrant/dev/eth_localdata/ --port 30307 console
```

eth01

```
geth --networkid 15150 --datadir ~/dev/eth_localdata/ --port 30303 --rpc --rpcport 8545 --rpcaddr 0.0.0.0 --rpccorsdomain "*" --rpcapi "admin,net,miner,eth,rpc,web3,txpool,debug,db,personal" console --allow-insecure-unlock 2>>
~/dev/eth_localdata/geth.log
```

- --allow-insecure-unlock 하면 unlock error 풀림

eth02

```
geth --networkid 15150 --maxpeers 5 --datadir /home/vagrant/dev/eth_localdata/ --port 30304 console 2>>
~/dev/eth_localdata/geth.log
```

eth03

```
geth --networkid 15150 --maxpeers 5 --datadir /home/vagrant/dev/eth_localdata/ --port 30305 console 2>>
~/dev/eth_localdata/geth.log
```

eth04

```
geth --networkid 15150 --maxpeers 5 --datadir /home/vagrant/dev/eth_localdata/ --port 30306 console 2>>
~/dev/eth_localdata/geth.log
```

eth05

```
geth --networkid 15150 --maxpeers 5 --datadir /home/vagrant/dev/eth_localdata/ --port 30307 console 2>>
~/dev/eth_localdata/geth.log
```

노드정보 확인

```
admin.nodeInfo.enode
```

```
"enode://c3250de76430e12e2b60aa96f97eb4bf1cbc7e8894e3c9fb632be989958f397c80f2bf9cd75114c97cabd7bd38e5ed2cc00b81c0aad40d2b52643a47702c4311
discport=51135"
```

노드연결

```
admin.addPeer ("노드주소")
```

ex

```
admin.addPeer
("enode://c3250de76430e12e2b60aa96f97eb4bf1cbc7e8894e3c9fb632be989958f397c80f2bf9cd75114c97cabd7bd38e5ed2cc00b81c0aad40d2b52643a47702c4311
discport=51135")
```

노드연결확인

```
admin.peers
net.peerCount
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 2: vagrant

> INFO [08-20|04:01:31.934] Regenerated local transaction journal transactions=0 accounts=0

> admin.peers
[
  {
    caps: ["eth/63"],
    enode: "enode://81e7f1a5820dfcfdcc0c882b05fcec9325433d51ff1d6c7a1b3e15e58db229cfb586e6e5c82230b661504d6647a3298e66c40ab38@192.168.50.12:41666",
    id: "0758281663c5594cf3fcf82f656da9eaea1761eaf4888be451fb14f2cdf41100",
    name: "Geth/v1.9.2-stable-e76047e9/linux-amd64/go1.11.5",
    network: {
      inbound: true,
      localAddress: "192.168.50.11:30303",
      remoteAddress: "192.168.50.12:41666",
      static: false,
      trusted: false
    }
  }
]
```

```
eth.sendTransaction({from: "0x63e9aea4c5e6bfd3a01fc53a13cad39c679fdb2d", to :
"0xb02d40a08ffcd7ecdb6ecf3b2b2096eaa137053b", value:web3.toWei(1,"ether")})
```

2019년 8월 26일 오후 2:30:22에서 수정 충돌 발생:

VSCode Terminal

PS c:\>로 바로 나가기

```
cd ../../
```

새로운 로컬 디렉토리 생성

```
mkdir ./dev/eth_prac001
```

```
디렉터리: C:\dev

Mode                LastWriteTime         Length Name
----                -
d-----          2019-08-19 오후 4:23             eth_prac001
```

파일 생성 확인하고 생성한 파일로 들어가기

```
ls
```

```
디렉터리: c:\

Mode                LastWriteTime         Length Name
----                -
d-----          2019-07-23 오전 11:15             bini
d-----          2019-08-19 오후 4:23             dev
d-----          2019-07-08 오전 10:13 errorwebmobileseconnd
d-----          2019-07-15 오후 5:56             googlekey
d-----          2019-06-25 오후 12:00             HashiCorp
d-----          2019-05-23 오후 9:15             Intel
d-----          2018-09-15 오후 4:33             PerfLogs
d-r---          2019-08-16 오후 7:19             Program Files
d-r---          2019-08-16 오후 3:43             Program Files (x86)
d-----          2019-08-19 오후 1:51             project_blockchain
d-----          2019-07-16 오전 9:35             study
d-----          2019-05-23 오후 8:46             Temp
d-----          2019-07-26 오후 4:46             test
d-r---          2019-07-24 오후 5:30             Users
d-----          2019-08-07 오후 7:33             UsersmulticampusAppData
d-----          2019-08-14 오후 2:58             vue_study
d-----          2019-08-14 오후 6:02             Windows
d-----          2019-07-05 오후 2:58             블록체인
d-----          2019-08-19 오후 1:30             새 폴더
-a----          2019-07-16 오후 5:45             184940971 atomfile.zip
-a----          2019-08-05 오후 5:56             157202356 final_webmobile.7z
-a----          2019-05-23 오후 9:12             3033 rhdssetup.log
-a----          2019-05-23 오후 9:15             189 Setup.log
-a----          2019-08-14 오후 3:32             21590669 vue_study.zip
```

생성된 파일로 들어와서 vagrant init
그리고 Vagrantfile 수정

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

# All Vagrant configuration is done below. The "2" in Vagrant.configure
# configures the configuration version (we support older styles for
# backwards compatibility). Please don't change it unless you know what
# you're doing.

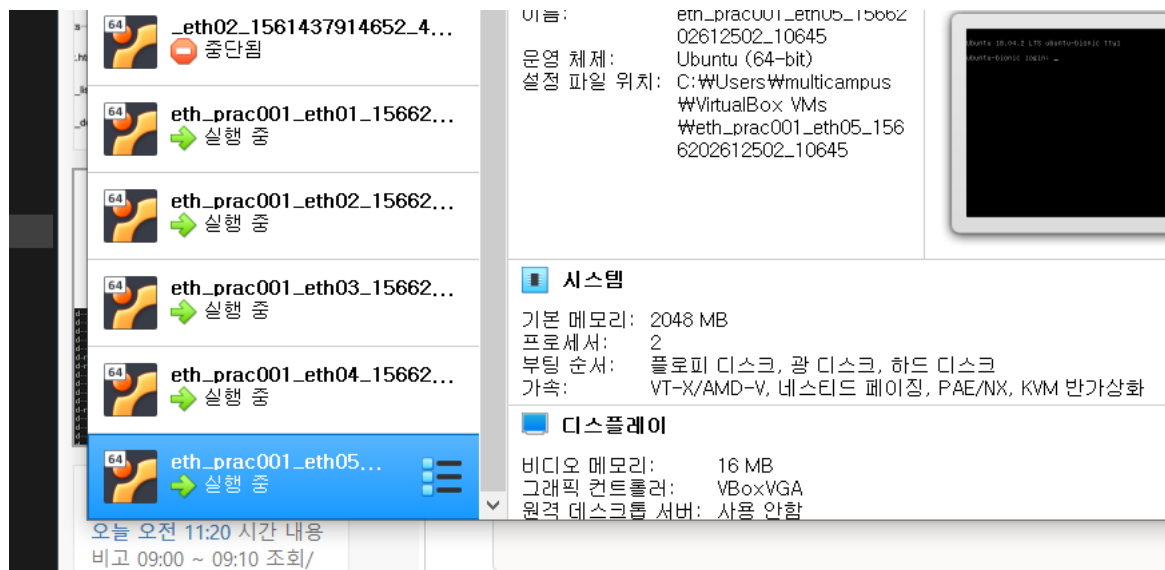
vms={
  eth01: ["10", 4096],
  eth02: ["11", 4096],
  eth03: ["12", 2048],
  eth04: ["13", 2048],
  eth05: ["14", 2048],
}

Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/bionic64"
  vms.map do |key, value|
    name = key.to_s
    ip_num, mem = value
    config.vm.define "#{name}" do |node|
      node.vm.network "private_network", ip: "192.168.50.#{ip_num}"
      node.vm.hostname = "#{name}"
      node.vm.provider "virtualbox" do |nodev|
        nodev.memory = "#{mem}"
      end
    end
  end
end
```

그리고 구동

```
vagrant up
```

그리고 virtualBox실행 시켜서 5개의 머신 확인하기



별도의 terminal에서 가상머신 접속 및 이더리움 설치 디렉토리로 들어가서

가상머신 접속하기

```
vagrant ssh eth01
```

가상머신에 `Geth` 설치

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

설치 후 설치 확인

```
geth version
```

```
vagrant@eth01:~$ geth version
INFO [08-20|00:42:20.584] Bumping default cache on mainnet
WARN [08-20|00:42:20.584] Sanitizing cache to Go's GC limits
Geth
Version: 1.9.2-stable
Git Commit: e76047e9f5499b58064bddde514dd3119a090adf
Architecture: amd64
Protocol Versions: [63]
Network Id: 1
Go Version: go1.11.5
Operating System: linux
GOPATH=
GOROOT=/usr/lib/go-1.11
vagrant@eth01:~$
```

가상머신 내 디렉토리 생성 및 GENESIS 블록파일 생성

```
mkdir -p dev/eth_localdata
cd dev/eth_localdata
vi CustomGenesis.json
```

point: vi 명령어 공부하기

a 키 누르고 붙여넣고 esc : wq하기

```
{
  "config": {
    "chainId": 15150,
    "homesteadBlock": 0,
    "eip155Block": 0,
    "eip158Block": 0
  },
  "difficulty": "0x10",
  "coinbase": "0x000000000000000000000000000000000000000000000000",
  "gasLimit": "9999999",
  "alloc": {},
  "extraData": "",
  "nonce": "0xdeadbeefdeadbeef",
  "mixhash": "0x000000000000000000000000000000000000000000000000",
  "parentHash": "0x000000000000000000000000000000000000000000000000",
  "timestamp": "0x00"
}
```

```
geth --datadir /home/vagrant/dev/eth_localdata/ init /home/vagrant/dev/eth_localdata/CustomGenesis.json
```

파일들 생성됐는지 확인

```
tree
```

트리설치(새로운 가상머신이기 때문에)

```
apt install tree
```

관리자로 설치해야 깔림

```
sudo apt install tree
```


Go Ethereum 구동

원래는

```
geth --networkid 15150 --maxpeers 5 --datadir /home/vagrant/dev/eth_localdata/ --port 30307 console
```

eth01

```
geth --networkid 15150 --datadir ~/dev/eth_localdata/ --port 30303 --rpc --rpcport 8545 --rpcaddr 0.0.0.0 --rpccorsdomain "*" --rpcapi "admin,net,miner,eth,rpc,web3,txpool,debug,db,personal" console --allow-insecure-unlock 2>>
~/dev/eth_localdata/geth.log
```

- --allow-insecure-unlock 하면 unlock error 풀림

eth02

```
geth --networkid 15150 --maxpeers 5 --datadir /home/vagrant/dev/eth_localdata/ --port 30304 console 2>>
~/dev/eth_localdata/geth.log
```

eth03

```
geth --networkid 15150 --maxpeers 5 --datadir /home/vagrant/dev/eth_localdata/ --port 30305 console 2>>
~/dev/eth_localdata/geth.log
```

eth04

```
geth --networkid 15150 --maxpeers 5 --datadir /home/vagrant/dev/eth_localdata/ --port 30306 console 2>>
~/dev/eth_localdata/geth.log
```

eth05

```
geth --networkid 15150 --maxpeers 5 --datadir /home/vagrant/dev/eth_localdata/ --port 30307 console 2>>
~/dev/eth_localdata/geth.log
```

노드정보 확인

```
admin.nodeInfo.enode
```

```
"enode://c3250de76430e12e2b60aa96f97eb4bf1cbc7e8894e3c9fb632be989958f397c80f2bf9cd75114c97cabd7bd38e5ed2cc00b81c0aad40d2b52643a47702c4311discport=51135"
```

노드연결

```
admin.addPeer ("노드주소")
```

ex

```
admin.addPeer
("enode://c3250de76430e12e2b60aa96f97eb4bf1cbc7e8894e3c9fb632be989958f397c80f2bf9cd75114c97cabd7bd38e5ed2cc00b81c0aad40d2b52643a47702c4311discport=51135")
```

노드연결확인

```
admin.peers
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 2: vagrant

> INFO [08-20|04:01:31.934] Regenerated local transaction journal transactions=0 accounts=0

> admin.peers
[{"caps": ["eth/63"],
  enode: "enode://81e7f1a5820dfcfdcc0c882b05fcec9325433d51ff1d6c7a1b3e15e58db229cfb586e6e5c82230b661504d6647a3298e66c40ab38@192.168.50.12:41666",
  id: "0758281663c5594cf3fcf82f656da9eaea1761eaf4888be451fb14f2cdf41100",
  name: "Geth/v1.9.2-stable-e76047e9/linux-amd64/go1.11.5",
  network: {
    inbound: true,
    localAddress: "192.168.50.11:30303",
    remoteAddress: "192.168.50.12:41666",
    static: false,
    trusted: false
  },
  url: "enode://81e7f1a5820dfcfdcc0c882b05fcec9325433d51ff1d6c7a1b3e15e58db229cfb586e6e5c82230b661504d6647a3298e66c40ab38@192.168.50.12:41666"
}]
```

```
eth.sendTransaction({from: "0x63e9aea4c5e6bfd3a01fc53a13cad39c679fdb2d", to :
"0xb02d40a08ffcd7ecdb6ecf3b2b2096eaa137053b", value:web3.toWei(1,"ether")})
```

03.블록체인 프로젝트_환경구성 가이드2

노트북: 블록체인

만든 날짜: 2019-08-21 오전 10:27

작성자: 비니문

업데이트:

2019-08-26 오후 2:47

문제점

Bootnode를 이용해 테스트 네트워크를 만들었을 때 초기화 바로 이후 Bootnode에 연결하는 잘 붙는데 geth를 종료하고 다시 Bootnode플래그로 연결하는 연결이 되지 않는 문제 발생

해결방법

일단 각 노드의 정보를 get

```
admin.nodeInfo.enode
```

eth01

```
"enode://c524d8705040793dfc89ab044cf9b68992c47557686270cd6b8d2928304437200111d26a7bb85bcc9ee4de81ce4ea2401d9d20b393330801f31bf1def76daf9cdiscport=59019"
```

eth02

```
"enode://c3250de76430e12e2b60aa96f97eb4b1cbc7e8894e3c9fb632be989958f397c80f2bf9cd75114c97cabd7bd38e5ed2cc00b81c0aad40d2b52643a47702c4311discport=64021"
```

eth03

```
"enode://81e7f1a5820dfcfdcc0c882b05fcec9325433d51ff1d6c7a1b3e15e58db229cfb586e6e5c82230b661504d6d77d93be50c078c9c8f4dd3647a3298e66c40ab3fdiscport=51024"
```

eth04

```
"enode://dbc1d41d33ed68b9e016e73fbaeadf0aee19389009155b9c416d490c940ccbcd9feb511a296d1dd675c8aa1a38b2c29ec234542130a0e5e0fd0b77d3129daf7discport=51026"
```

eth05

```
"enode://73a7f537b0c70038896766a54c4656948a703ab7c634b75b388001b54b9e308c9aa7252132fcc8bf45240de8a0c5c33a6ba2b367383448c173c9bfa6207d3748discport=51027"
```

cd dev/eth_localdata에 들어가서

vi static-nodes.json 생성하고

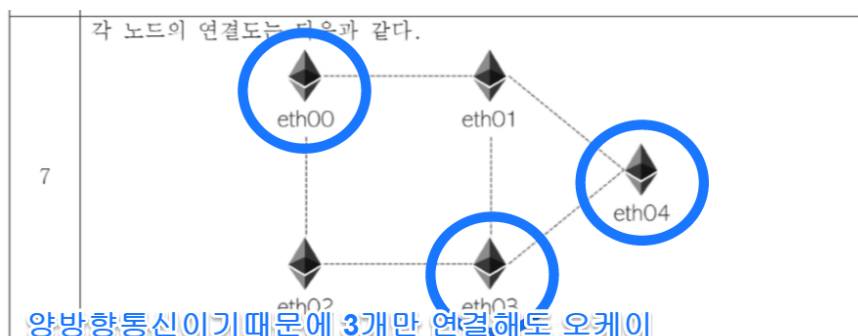
```
["enode://c3250de76430e12e2b60aa96f97eb4b1cbc7e8894e3c9fb632be989958f397c80f2bf9cd75114c97cabd7bd38e5ed2cc00b81c0aad40d2b52643a47702c4311discport=64021", "enode://81e7f1a5820dfcfdcc0c882b05fcec9325433d51ff1d6c7a1b3e15e58db229cfb586e6e5c82230b661504d6d77d93be50c078c9c8f4dd3647a3298e66c40ab3fdiscport=51024"]
```

```
["enode://c524d8705040793dfc89ab044cf9b68992c47557686270cd6b8d2928304437200111d26a7bb85bcc9ee4de81ce4ea2401d9d20b393330801f31bf1def76daf9cdiscport=59019", "enode://dbc1d41d33ed68b9e016e73fbaeadf0aee19389009155b9c416d490c940ccbcd9feb511a296d1dd675c8aa1a38b2c29ec234542130a0e5e0fd0b77d3129daf7discport=51026", "enode://73a7f537b0c70038896766a54c4656948a703ab7c634b75b388001b54b9e308c9aa7252132fcc8bf45240de8a0c5c33a6ba2b367383448c173c9bfa6207d3748discport=51027"]
```

```
["enode://c524d8705040793dfc89ab044cf9b68992c47557686270cd6b8d2928304437200111d26a7bb85bcc9ee4de81ce4ea2401d9d20b393330801f31bf1def76daf9cdiscport=59019", "enode://dbc1d41d33ed68b9e016e73fbaeadf0aee19389009155b9c416d490c940ccbcd9feb511a296d1dd675c8aa1a38b2c29ec234542130a0e5e0fd0b77d3129daf7discport=51026"]
```

```
["enode://c3250de76430e12e2b60aa96f97eb4b1cbc7e8894e3c9fb632be989958f397c80f2bf9cd75114c97cabd7bd38e5ed2cc00b81c0aad40d2b52643a47702c4311discport=64021", "enode://81e7f1a5820dfcfdcc0c882b05fcec9325433d51ff1d6c7a1b3e15e58db229cfb586e6e5c82230b661504d6d77d93be50c078c9c8f4dd3647a3298e66c40ab3fdiscport=51024", "enode://73a7f537b0c70038896766a54c4656948a703ab7c634b75b388001b54b9e308c9aa7252132fcc8bf45240de8a0c5c33a6ba2b367383448c173c9bfa6207d3748discport=51027"]
```

```
["enode://c3250de76430e12e2b60aa96f97eb4b1cbc7e8894e3c9fb632be989958f397c80f2bf9cd75114c97cabd7bd38e5ed2cc00b81c0aad40d2b52643a47702c4311discport=64021", "enode://dbc1d41d33ed68b9e016e73fbaeadf0aee19389009155b9c416d490c940ccbcd9feb511a296d1dd675c8aa1a38b2c29ec234542130a0e5e0fd0b77d3129daf7discport=51026"]
```



Go Ethereum 구동

eth01

```
geth --networkid 15150 --datadir ~/dev/eth_localdata/ --port 30303 --rpc --rpcport 8545 --rpcaddr 0.0.0.0 --rpccorsdomain "*" --rpcapi "admin,net,miner,eth,rpc,web3,txpool,debug,db,personal" console
```

eth02

```
geth --networkid 15150 --maxpeers 5 --datadir /home/vagrant/dev/eth_localdata/ --port 30307 console
```

연결 확인 체크

```
net.peerCount
```

04.블록체인 프로젝트_AWS 가이드

노트북: 블록체인
만든 날짜: 2019-08-26 오후 2:47
작성자: 비니문
URL: http://54.180.138.195/server.html

업데이트: 2019-10-21 오후 3:44

PutTY

- 가상 단말기 프로그램

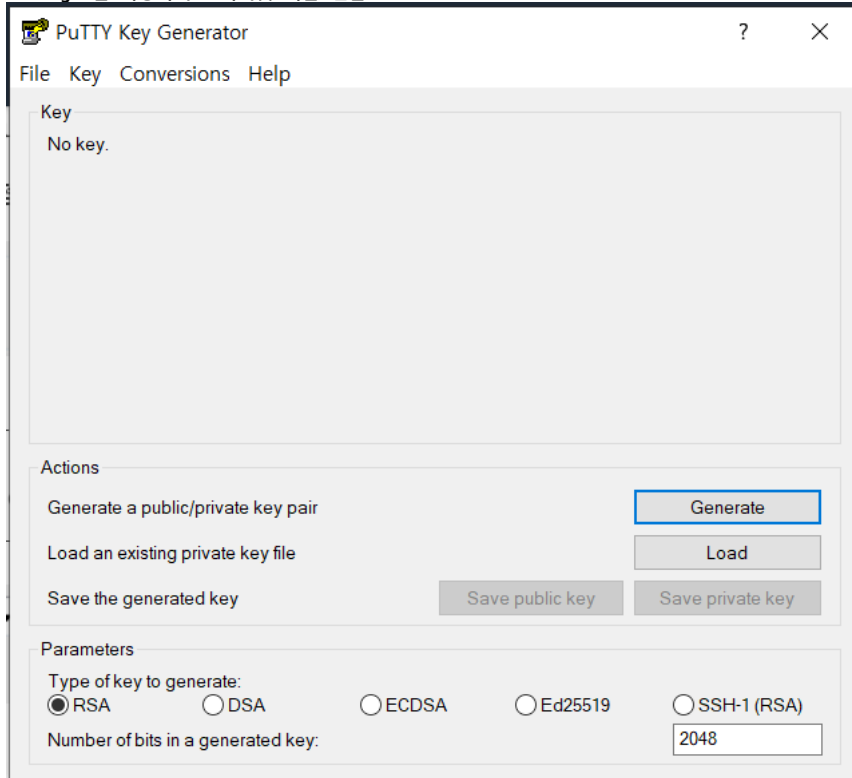
aws 가이드

https://docs.aws.amazon.com/ko_kr/AWSEC2/latest/UserGuide/putty.html

PutTY 다운로드

<https://www.chiark.greenend.org.uk/~sgtatham/putty/>

PutTYgen을 사용하여 프라이빗 키를 변환

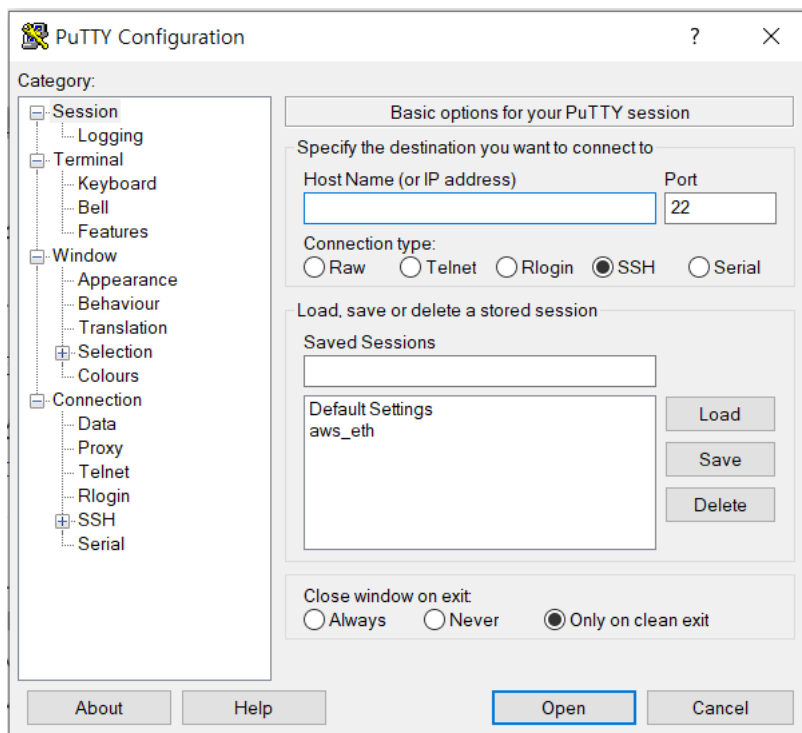


- 키 유형: RSA
- generated key: 2048로 설정하기
- Load에서 key 선택(.pem)
- Save private key(.ppk)로 key 변환 성공

PutTY 사용하여 인스턴스에 연결

http ip 아는 사이트

<http://54.180.138.195/server.html>



- Host Name

user_name@public_dns_name
예) ubuntu@54.180.162.22

- Connection/SSH/Auth
 - private key file for authentication에서 키(.ppk) 선택하기
- Connection
 - 여기서 시간 설정해주기 나는 180으로 설정함
- Session
 - Saved Sessions이름 설정하고 Save 하면 됨
 - ex)aws_eth

WinSCP 다운로드

<http://winscp.net/eng/download.php>

- 호스트 이름 (Host name)
 - 인스턴스 퍼블릭 DNS 호스트 이름
 - 퍼블릭 IPv4 주소를 입력

54.180.162.22

- 사용자 이름: ubuntu
- 비밀번호
 - 고급에서 선택
 - 인증/개인키 파일: 아까 생성한 key 넣기

그러면 서버 접속 성공!!

간단 접속법

```
touch (key_name).pem
vi (key_name).pem
chmod 400 (key_name).pem
ssh -i (key_name).pem ubuntu@(public ip)
```

05.mining 과정

노트북:	블록체인	업데이트:	2019-10-02 오전 10:20
만든 날짜:	2019-08-21 오후 2:49		
작성자:	비니문		
URL:	chrome-extension://nkbihfbeogaeaoehlefnkodbefgpgknn/popup.html		

계정생성

```
personal.newAccount("계정")
```

계정확인

```
eth.accounts  
  
["0xc6c78407a51eda1ac521047185748f41ca43315d"]
```

채굴시 보상받는 계정을 지정시

```
miner.setEtherbase(personal.listAccounts[0])
```

채굴시작

```
miner.start(1)
```

채굴 여부 확인

```
eth.mining
```

채굴종료

```
miner.stop()
```

획득한 보상 확인

```
web3.fromWei(eth.getBalance(eth.accounts[0]))
```

획득한 이더 확인

```
web3.fromWei(eth.getBalance(eth.accounts[0]))
```

unlock이 걸려있다면 먼저 풀고주고

```
web3.personal.unlockAccount(eth.coinbase, "계정")  
  
web3.personal.unlockAccount(eth.coinbase)  
  
personal.unlockAccount("주소")
```

이더 전송하기

```
eth.sendTransaction({from: "", to: "", value: web3.toWei(보내고싶은이더수, "ether")})
```

그리고 시작.종료 다시해주기

```
miner.start()  
miner.stop()
```

생성된 블록 수 조회

```
eth.blockNumber
```

n번째 블록의 정보 출력

```
eth.getBlock(n)
```

계정의 상태를 출력(Locked or Unlocked)

```
personal.listWallets[0].status
```

처리해야 할 트랜잭션 목록

```
eth.pendingTransactions
```

해쉬에 대한 목록 확인

```
eth.getTransaction("0x262f16384114fd3f31d3dcb44176c2d46b2b3f4d22d34dae2f5036ddba8a73b7")
```

```
eth.sendTransaction({from: "0xc6c78407a51eda1ac521047185748f41ca43315d", to: "0x449D06180B14cEB85652AA1bE979ec5e7b38D8fE",  
value: web3.toWei(1, "ether")})
```

06. geth.start

노트북:	블록체인	업데이트:	2019-09-09 오전 11:18
만든 날짜:	2019-08-26 오후 4:49		
작성자:	비니문		
URL:	https://ko.wikipedia.org/wiki/Nohup		

01

```
geth --networkid 15150 --datadir ~/Ethereum1/dev/eth_localdata --port 3333 --rpc --rpcport 8545 --rpcaddr 0.0.0.0 --rpccorsdomain "*" --rpcapi "admin,net,miner,eth,_rpc,web3,txpool,debug,db,personal" console 2>> ~/Ethereum1/dev/eth_localdata/geth.log
```

이더리움 충전할 때 --allow-insecure-unlock을 써줘야 가능함

```
geth --networkid 15150 --datadir ~/Ethereum1/dev/eth_localdata --port 3333 --rpc --rpcport 8545 --rpcaddr 0.0.0.0 --rpccorsdomain "*" --rpcapi "admin,net,miner,eth,_rpc,web3,txpool,debug,db,personal" --allow-insecure-unlock console 2>> ~/Ethereum1/dev/eth_localdata/geth.log
```

```
nohup geth --networkid 15150 --datadir ~/Ethereum1/dev/eth_localdata --port 3333 --rpc --rpcport 8545 --rpcaddr 0.0.0.0 --rpccorsdomain "*" --rpcapi "admin,net,miner,eth,_rpc,web3,txpool,debug,db,personal" & console 2>> ~/Ethereum1/dev/eth_localdata/geth.log
```

nohup예시

```
$ nohup abcd &  
$ exit
```

아래의 명령 중 첫 번째 항목은 백그라운드에서 abcd라는 프로그램을 시작한다. 그 뒤 로그아웃을 하더라도 이 프로그램의 프로세스를 중단하지는 않는다

```
nohup geth --networkid 15150 --datadir ~/Ethereum1/dev/eth_localdata --port 3333 & console 2>> ~/Ethereum1/dev/eth_localdata/geth.log
```

최종

```
geth --networkid 15150 --datadir ~/Ethereum1/dev/eth_localdata --port 3333 --rpc --rpcport 8545 --rpcaddr 0.0.0.0 --rpccorsdomain "*" --rpcapi "admin,net,miner,eth,_rpc,web3,txpool,debug,db,personal" --allow-insecure-unlock console 2>> ~/Ethereum1/dev/eth_localdata/geth.log
```

02

```
geth --networkid 15150 --maxpeers 5 --datadir ~/Ethereum2/dev/eth_localdata/ --port 3334 console 2>> ~/Ethereum2/dev/eth_localdata/geth.log
```

03

```
geth --networkid 15150 --maxpeers 5 --datadir ~/Ethereum3/dev/eth_localdata/ --port 3335 console 2>> ~/Ethereum3/dev/eth_localdata/geth.log
```

일반적 검색
ps

```
"geth"들어가는 프로그램 검색  
-ef | grep geth
```

07. 01 Docker란?

노트북: 블록체인

만든 날짜: 2019-08-27 오전 9:33

작성자: 비니문

업데이트:

2019-08-29 오전 9:36

도커(Docker)

- 리눅스 컨테이너를 기반으로 하여 특정한 서비스를 패키징하고 배포하는데 유용한 오픈소스 프로그램

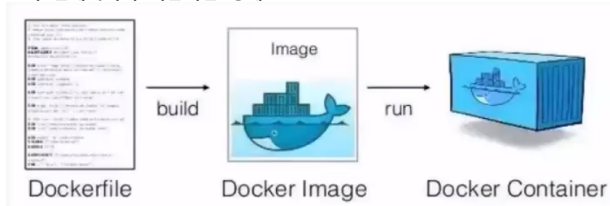
배포 도구

- Maven, Gradle, FTP...
- 이러한 배포 도구들의 사용이유
 - 특정한 소프트웨어를 개발한 이후에 배포하자고 할 때 해당 프로그램이 어디에선가 손쉽게 구동이 되도록 하기 위해서 사용

도커의 컨테이너(Container)

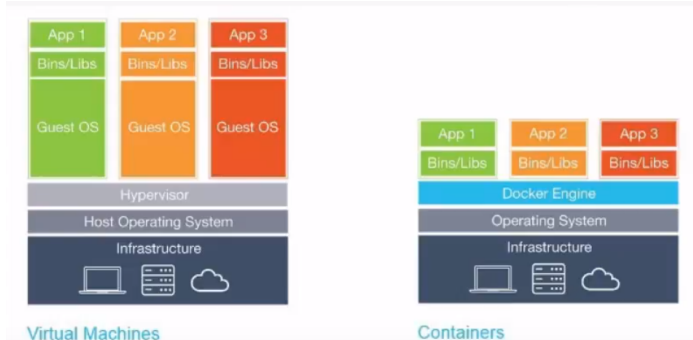
- 도커를 설치하면 '이미지(Image)'를 담아서 구동시키는 방식으로 손쉽게 배포
- 도커를 설치할 하면 나머지를 알아서 설정이 이루어짐
 - JVM, tomcat 등을 따로 설치할 필요가 X, 도커 하나만 있으면 해결됨=>간략화됨
- 통일된 형태로 배포는 쉽게

도커 컨테이너의 기본적인 형태



- 도커 파일 생성: '어떠한 소프트웨어를 컨테이너에 담아서 구동 시킬 것인지를 명세'
- bulid 해주면 알아서 도커 이미지가 도커 파일에 맞게 생성
- 도커 이미지는 run하면 순식간에 도커 컨테이너 위에서 실행 됨

VM vs 도커 컨테이너



- VM
 - 가상화 기능을 사용해 Guest OS라는 것을 만들어냄
 - Guest OS와 Host OS는 별개의 존재 => 서로 의존적이지 않음
 - Guest OS속도가 느리고 용량이 크다는 단점이 발생
- 컨테이너(Container)
- 도커 엔진 위에서 동작 => 별도의 Guest OS가 사용되지 않아 성능적으로 매우 개선
- 메모리 용량도 적게 차지
- 다만 기본적으로 사용하는 있는 운영체제와 도커의 컨테이너에 의존성이 존재함

07.02 Docker설치과정

노트북: 블록체인

만든 날짜: 2019-08-29 오전 9:30

업데이트:

2019-08-29 오전 9:36

작성자: 비니문

도커명령어

```
sudo docker ps      -> 현재 실행하고 있는 컨테이너 확인
sudo docker ps -a    -> 내 모든 컨테이너 확인
docker version       -> 버전확인
```

도커설치

```
sudo passwd root -> 초기 비밀번호 설정
sudo apt update -> 업데이트 한번 하기
```

<일반적인 설치>

```
sudo apt install apt-transport-https
sudo apt install ca-certificates
sudo apt install curl
sudo apt install software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable"
sudo apt update
```

<도커 설치>

```
apt-cache policy docker-ce
sudo apt install docker-ce
```

<도커 상태확인>

```
sudo systemctl status docker
```

<test>

```
sudo docker run hello-world
//이미지 씬 없으면 다운받아와서 가져옴
docker images ->내가 가진 이미지 확인
```

<실행>

```
sudo docker run hello-world
어떤 컨테이너 삭제
docker rm 이미지이름
```

<도커실행할때>

```
docker run -p 호스트port:컨테이너포트 레포지토리이름
```

<컨테이너 모조리 강제 삭제>

```
docker rm -f $(docker ps -a -q)
```

<마운트 시키는법>

```
docker run -p 호스트port:컨테이너포트 -v /home/ubuntu/경로/:/경로 레포지토리이름
```

<mysql>

```
도커이미지 다운로드
docker pull 도커허브 이름
docker pull emblockit/haribo-mysql
```

<실행>

```
docker run -d -p 3307:3306 --name vilien-mysql -v /home/ubuntu/chainVilien/mysql:/var/lib/mysql emblockit/haribo-mysql
```

<mysql 접속>

```
docker exec -it 862c5f4b8701 /bin/bash
```

<그냥 밖에서 이거 입력하면 mysql 접속 ㄴ>

```
mysql -u root -p --host 127.0.0.1 --port 3307
```

<컨테이너로 접속하면>

```
mysql -u root -p
패스워드입력
Id: haribo // pw : haribo
root // hariboadmin
```

<컨테이너정보보기>

```
docker inspect 컨테이너아이디
```

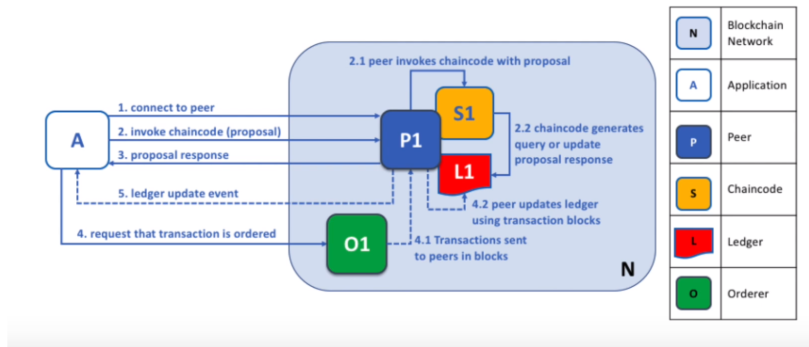

08. 하이퍼 레저 패브릭

노트북: 블록체인
 만든 날짜: 2019-09-02 오후 11:12
 작성자: 비니문

업데이트: 2019-09-02 오후 11:32

Fabric 구조

그림이해중요



비트코인, 이더리움과 같은점과 다른점

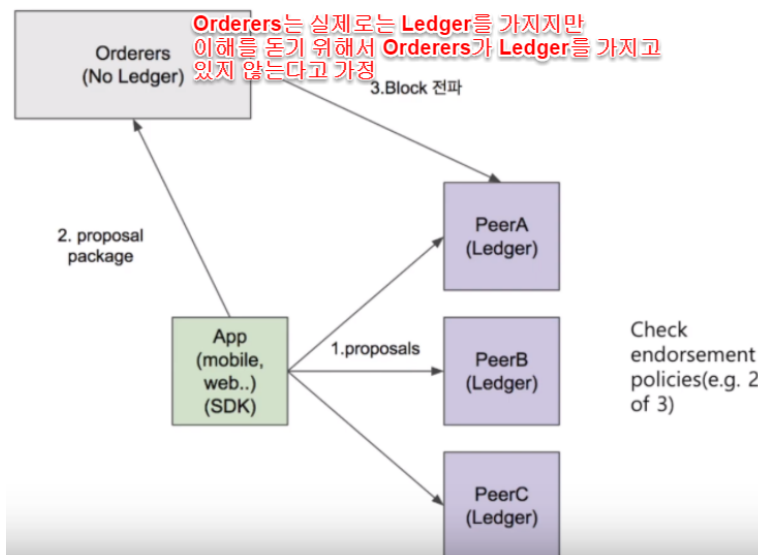
- 비트코인, 이더리움: public blockchain
 - 블록 생성자 == 스마트 컨트랙트 처리 노드
 - 개인키 기반
 - 개인적으로 private key를 만들어서 그 기반으로 돈을 주고받고 함
- Fabric
 - 블록생성자 != 스마트 컨트랙트 처리 노드
 - Orderer = 블록 생성만
 - Peer = 시뮬레이션, 블록체인 보관
 - 인증서 기반
 - 은행에서 발급받은 공인인증서 사용하듯이 중간관리자가 있음
 - Fabric CA라는 중간 관리자(Certification authority)
 - 중간관리자가 인증서를 발급해주고 패브릭은 이 인증서를 기반으로 활동
 - 하나의 키 값을 한 블록에서 여러번 바꿀 수가 없음
 - 일반적으로 생각하는 방식으로는 X
 - 한 블록에서 A=100이었다 => A += 100 => A == 200
 - 위와 같은 방식은 비트코인과 이더리움에서는 가능하나 패브릭에서 불가능
 - 패브릭에서 불가능한 이유는 패브릭의 아키텍처 때문에 발생하는 가장 큰 차이점
- **패브릭과 VS 비트코인, 이더리움의 가장 큰 차이점은?**

비트코인과 이더리움의 경우 프로그램 하나가 모든 것을 처리함
 패브릭은 프로그램 하나가 아니라 구성요소가 나누어져 있기 때문에 각 역할이 나누어져 있음

비트코인과 이더리움은 Geth를 실행해서 마이닝, 지갑연결, 블록체인 저장, 월드스테이트 가지고 있어서 프로그램 하나로 모든 역할이 다 가능
 패브릭은 Orderer= 블록생성, Peer= 시뮬레이션&블록체인 보관 이런식으로 나누어져 있음

구성요소

- Peer
 - 블록체인 보관(Ledger: 안에 World State 보관)
 - 거래 트랜잭션 시뮬레이션
- Orderer
 - 블록을 생성
 - 거래(트랜잭션)을 모아서 블록을 생성만 하고 시뮬레이션은 X
- App(유저)
 - 인증서/지갑을 가지고 거래를 만들고 Peer, Orderer와 상호작용



- 유저(App)가 거래를 만들
- 1.proposals(endorsement policies)

- 유저가 Peer들에게 거래를 보냄
- 이 거래는 블록에 바로 들어가는 게 아니라 제안
- Peer는 이 거래를 시뮬레이팅함(이 거래가 어떨지~)
- Peer는 이 거래가 괜찮으면 OK를 보냄 --> 패브릭에서의 합의방식(투표)
 - BET
 - 다수결_3명 중 2명 이상이 OK하면 됨
 -

08.01 하이퍼 레저 패브릭

노트북: 블록체인

만든 날짜: 2019-09-02 오후 2:51

업데이트: 2019-09-03 오전 11:04

작성자: 비니문

URL: <https://medium.com/decipher-media/%ED%95%98%EC%9D%B4%ED%8D%BC%EB%A0%88%EC%A0%80-%ED%8C%A8%EB%B8%8C%EB%A6%...>

하이퍼레저 패브릭과 다른 블록체인 프로젝트들과의 비교

	비트코인	이더리움	하이퍼레저 패브릭
자체 화폐	비트코인	이더	없음
네트워크	공개	공개형	허가형
거래	익명	익명 또는 비공개 (지원 예정)	공개 또는 기밀
합의	POW	POW, POS (지원 예정)	SOLO Kafka, PBFT (v1.0 이전) SBFT (지원 예정)

하이퍼레저 패브릭의 특징

- 허가형 블록체인으로서는 허가 받은 참여자만 네트워크에 참여할 수 있다.
- 스마트 컨트랙트에 일반 프로그래밍 언어 사용이 가능하다. (현재는 go, Node.js 지원)
- 스마트 컨트랙트를 일부 노드만 실행하므로 다수의 거래를 병렬적으로 빠르게 처리할 수 있다.
- 채 널을 이용해 허가 받은 사람들에게만 장부(ledger)를 공개할 수 있다.
- 교제 가능한 합의 프로토콜을 사용할 수 있다. (SOL0, Kafka 방식, PBFT등)
- 허가형 블록체인으로서는 네트워크 참여자의 신원을 확인할 수 있기 때문에 문제 발생시 책임소재를 분명히 할 수 있다.

하이퍼레저

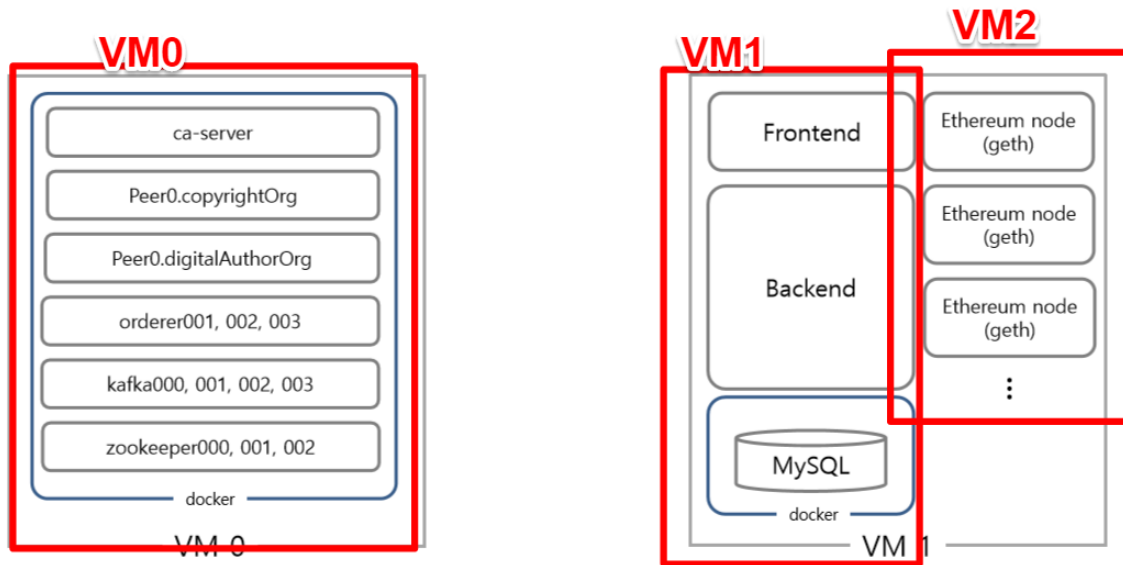
09. 하이퍼 레저 패브릭 설치

노트북: 블록체인
만든 날짜: 2019-09-02 오전 11:20
작성자: 비니문

업데이트: 2019-09-12 오후 4:00

Virtual box ubuntu 세팅하기

네트워크 구조



주요 디렉토리 및 파일



초기 비밀번호 설정

```
sudo passwd root
```

도커 설치

```
sudo apt updated
sudo apt install apt-transport-https
sudo apt install ca-certificates
sudo apt install curl
sudo apt install software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable"
sudo apt update
apt-cache policy docker-ce
sudo apt install docker-ce
```

도커 버전확인

```
sudo systemctl status docker
```

도커 컴포즈

```
sudo curl -L "https://github.com/docker/compose/releases/download/1.24.1/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

sudo chmod +x /usr/local/bin/docker-compose
```

네트워크 구동 순서

- 전체 순서
 - kafka-zookeeper cluster 실행 -> 채널 구성요소 생성 -> orderer실행 -> peer 실행 -> 채널 생성 및 참여 ->앵커 피어 업데이트
 - 모든 작업은 AuctionConsortium 디렉토리 내부에서 진행
 - 모든 명령어는 command-line.sh 파일에 저장되어 있음

Kafka-zookeeper cluster 실행

```
sudo docker-compose -f zookeeper-compose.yaml up -d
sudo docker-compose -f kafka-compose.yaml up -d
```

권한주기

```
chmod 777 configtxgen
```

채널 구성요소 생성

- 패브릭 채널 구성을 위한 트랜잭션(~.tx)와 Genesis block(~.block) 생성

```
./configtxgen -profile AuctionOrdererGenesis -outputBlock orderer-genesis.block -channelID ordererchannel001

./configtxgen -profile AuctionChannel -outputCreateChannelTx auctionchannel001.tx -channelID auctionchannel001
```

생성된 트랜잭션 파일 이동

```
mv auctionchannel001.tx ./channel-artifacts
```

Orderer 실행

- 관련 파일 디렉토리:AuctionConsortium/
 - order-compose.yaml

```
sudo docker-compose -f orderer-compose.yaml up -d
```

peer 실행

```
vi peer-compose.yaml
```

- 구동 전에, CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE 값 확인 혹은 변경
 - peer-compose.yaml 파일이 속해있는 디렉토리가 AuctionConsortium일 경우, auctionconsortium_default으로 할당(반드시 소문자)
 - <directory_name>_default [2군데있음]

```
sudo docker-compose -f peer-compose.yaml up -d
```

채널 생성

```
sudo docker exec -e "CORE_PEER_LOCALMSPID=copyrightAssociateOrgMSP" -e "CORE_PEER_MSPCONFIGPATH=/var/hyperledger/users/msp" peer0.copyrightOrg peer channel create -o orderer001:7050 -c auctionchannel001 -f /var/hyperledger/production/auctionchannel001.tx --outputBlock /var/hyperledger/production/auctionchannel001-genesis.block
```

채널 참여

```
sudo docker exec -e "CORE_PEER_LOCALMSPID=copyrightAssociateOrgMSP" -e "CORE_PEER_MSPCONFIGPATH=/var/hyperledger/users/msp" peer0.copyrightOrg peer channel fetch config /var/hyperledger/production/auctionchannel001-genesis.block -o orderer001:7050 -c auctionchannel001
```

```
sudo docker exec -e "CORE_PEER_LOCALMSPID=copyrightAssociateOrgMSP" -e "CORE_PEER_MSPCONFIGPATH=/var/hyperledger/users/msp" peer0.copyrightOrg peer channel join -o orderer001:7050 -b /var/hyperledger/production/auctionchannel001-genesis.block
```

```
sudo docker exec -e "CORE_PEER_LOCALMSPID=digitalAuthorOrgMSP" -e "CORE_PEER_MSPCONFIGPATH=/var/hyperledger/users/msp" peer0.digitalAuthorOrg peer channel fetch config /var/hyperledger/production/auctionchannel001-genesis.block -o orderer001:7050 -c auctionchannel001
```

```
sudo docker exec -e "CORE_PEER_LOCALMSPID=digitalAuthorOrgMSP" -e "CORE_PEER_MSPCONFIGPATH=/var/hyperledger/users/msp" peer0.digitalAuthorOrg peer channel join -o orderer001:7050 -b /var/hyperledger/production/auctionchannel001-genesis.block
```

앵커 피어 업데이트 트랜잭션 생성

```
./configtxgen -profile AuctionChannel -outputAnchorPeersUpdate copyrightOrgAnchor.tx -channelID auctionchannel001 -asOrg copyrightAssociateOrgMSP
```

```
./configtxgen -profile AuctionChannel -outputAnchorPeersUpdate digitalAuthorOrgAnchor.tx -channelID auctionchannel001 -asOrg digitalAuthorOrgMSP
```

앵커 피어 업데이트 트랜잭션 이동

```
mv copyrightOrgAnchor.tx ./channel-artifacts
mv digitalAuthorOrgAnchor.tx ./channel-artifacts
```

앵커 피어 업데이트

```
sudo docker exec -e "CORE_PEER_LOCALMSPID=copyrightAssociateOrgMSP" -e "CORE_PEER_MSPCONFIGPATH=/var/hyperledger/users/msp" peer0.copyrightOrg peer channel update -o orderer001:7050 -c auctionchannel001 -f /var/hyperledger/production/copyrightOrgAnchor.tx
```

```
sudo docker exec -e "CORE_PEER_LOCALMSPID=digitalAuthorOrgMSP" -e "CORE_PEER_MSPCONFIGPATH=/var/hyperledger/users/msp"
peer0.digitalAuthorOrg peer channel update -o orderer001:7050 -c auctionchannel001 -f
/var/hyperledger/production/digitalAuthorOrgAnchor.tx
```

체인코드 설치 및 배포

체인코드 설치(구동 확인을 위하여 체인코드가 아닌 완성본을 설치)

- Nodejs체인코드위치: AuctionConsortium/chaincode/node/AssetManagementjs
- 관련파일 :AuctionConsortium/chaincode/node/package.json

```
sudo docker exec -e "CORE_PEER_LOCALMSPID=copyrightAssociateOrgMSP" -e "CORE_PEER_MSPCONFIGPATH=/var/hyperledger/users/msp"
peer0.copyrightOrg peer chaincode install -n asset -v 0.1 -l node -p /var/hyperledger/chaincode/node
```

```
sudo docker exec -e "CORE_PEER_LOCALMSPID=digitalAuthorOrgMSP" -e "CORE_PEER_MSPCONFIGPATH=/var/hyperledger/users/msp"
peer0.digitalAuthorOrg peer chaincode install -n asset -v 0.1 -l node -p /var/hyperledger/chaincode/node
```

체인코드 인스턴스 생성(체인코드 도커 컨테이너를 생성 및 구동)

```
sudo docker exec -e "CORE_PEER_LOCALMSPID=digitalAuthorOrgMSP" -e "CORE_PEER_MSPCONFIGPATH=/var/hyperledger/users/msp"
peer0.digitalAuthorOrg peer chaincode instantiate -o orderer001:7050 -C auctionchannel001 -n asset -v 0.1 -c '{"Args":["init"]}' -P
"OR('copyrightAssociateOrgMSP.member','digitalAuthorOrgMSP.member')"
```

생성된 체인코드 동작 테스트

- AssetMangementjs의 registerAsset 함수

```
sudo docker exec -e "CORE_PEER_LOCALMSPID=digitalAuthorOrgMSP" -e "CORE_PEER_MSPCONFIGPATH=/var/hyperledger/users/msp"
peer0.digitalAuthorOrg peer chaincode invoke -o orderer001:7050 -C auctionchannel001 -n asset -c '{"Args":
["registerAsset","testasset","testowner"]}'
```

- AssetManagementjs의confirmTimestamp함수

```
docker exec -e "CORE_PEER_LOCALMSPID=digitalAuthorOrgMSP" -e "CORE_PEER_MSPCONFIGPATH=/var/hyperledger/users/msp"
peer0.digitalAuthorOrg peer chaincode invoke -o orderer001:7050 -C auctionchannel001 -n asset -c '{"Args":["
confirmTimestamp","testasset"]}'
```

체인코드의 소스코드 내용이 변경 될 경우 수행

- 업그레이드를 위한 체인코드 패키지 생성
 - 수행 시 체인코드 버전을 올려 배포해야 함
 - -v 플래그에 업그레이드 된 버전을 입력

```
docker exec -e "CORE_PEER_LOCALMSPID=digitalAuthorOrgMSP" -e "CORE_PEER_MSPCONFIGPATH=/var/hyperledger/users/msp"
peer0.digitalAuthorOrg peer chaincode upgrade -o orderer001:7050 -C auctionchannel001 -n asset -v 0.2 -c '{"Args":["init"]}' -P
"OR('copyrightAssociateOrgMSP.member','digitalAuthorOrgMSP.member')"
```

생성된 체인코드를 날려 invoke 시 등록된 정보가 반환되며 정상동작

```
sudo docker exec -e "CORE_PEER_LOCALMSPID=digitalAuthorOrgMSP" -e "CORE_PEER_MSPCONFIGPATH=/var/hyperledger/users/msp"
peer0.digitalAuthorOrg peer chaincode query -o orderer001:7050 -C auctionchannel001 -n asset -c '{"Args":["query","testasset"]}'
```

이전 버전 체인코드 삭제

- 구동 중인 도커 컨테이너들 중 이전 버전 체인코드의 도커 컨테이너 확인

```
docker ps -a
```

- 이번 저번 체인코드 도커 컨테이너의 ID stop하고 삭제

```
docker stop <이전버전 체인코드 도커 컨테이너 ID>
docker rm <이전버전 체인코드 도커 컨테이너 ID>
```

- 이후 아래 파일들을 삭제

AuctionConsortium/copyrightOrg/ledger/chaincodes/ 에있는이전버전파일삭제 AuctionConsortium/digitalAuthorOrg/ledger/chaincodes/에있는이전
버전파일삭제

패브릭 네트워크 연동 가이드

- 패브릭 네트워크 <=> 백엔드 간 연동
 - 백엔드 측에서 아래와 같이 연동 정보 수정
 - 수정할파일:haribo-backend/src/main/resources/application.properties

변수명	값	비고
fabric.ca-server.url	http://xxx.xxx.xxx.xxx:8054	클라우드 vm에 할당된 공인 IP 입력
fabric.ca-server.admin.name	admin	기존 값에서 변경
fabric.ca-server.pem.file	fabric-ca.pem	AuctionConsortium/certificate-authority/CA_SERVER_HOME/ca-cert.pem 파일의 내용을 복사
fabric.org.name	digitalAuthorOrg	기존 값에서 변경
fabric.org.msp.name	digitalAuthorOrgMSP	기존 값에서 변경
fabric.org.admin.name	admin@digitalAuthorOrg	기존 값에서 변경
fabric.org.user.name	admin@digitalAuthorOrg	기존 값에서 변경
fabric.org.user.secret	pwd	기존 값에서 변경
fabric.peer.name	peer0.digitalAuthorOrg	기존 값에서 변경
fabric.peer.url	grpc://xxx.xxx.xxx.xxx:8051	클라우드 vm에 할당된 공인 IP 입력
fabric.peer.pem.file	fabric-peer.pem	AuctionConsortium/digitalAuthorOrg/peer0.digitalAuthorOrg/msp/cacert/0-0-0-0-7054.pem 파일의 내용을 복사
fabric.orderer.name	orderer001	기존 값에서 변경
fabric.orderer.url	grpc://xxx.xxx.xxx.xxx:8050	클라우드 vm에 할당된 공인 IP 입력
fabric.orderer.pem.file	fabric-orderer.pem	AuctionConsortium/default-ordering-service/orderer/orderer001/msp/cacert/0-0-0-0-7054.pem 파일의 내용을 복사
fabric.channel.name	auctionchannel001	기존 값에서 변경

- 백엔드 측 정보 수정 후 CA 서버 등록
 - 관련파일위치:AuctionConsortium/certificate-authorith/
 - 관련파일: ca-compose.yaml

```
docker-compose -f ./certificate-authority/ca-compose.yaml up -d
```

10. 01 하이퍼패브릭 네트워크 및 체인코드 배포

노트북:	블록체인	업데이트:	2019-09-09 오전 11:11
만든 날짜:	2019-09-09 오전 11:05		
작성자:	비니문		

초기 root 비밀번호 설정

```
sudo passwd root
```

```
sudo apt update
sudo apt install apt-transport-https
sudo apt install ca-certificates
sudo apt install curl
sudo apt install software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable"
sudo apt update
apt-cache policy docker-ce
sudo apt install docker-ce
```

도커 버전확인

```
sudo systemctl status docker
```

도커 컴포즈 다운로드

```
sudo curl -L "https://github.com/docker/compose/releases/download/1.24.1/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

Docker Compose가 위치한 /usr/local/bin/docker-compose 경로에 실행 권한을 추가

```
sudo chmod +x /usr/local/bin/docker-compose
```

컴퓨터에 Auction폴더 복붙하고 폴더로 이동

```
sudo docker-compose -f zookeeper-compose.yaml up -d
sudo docker-compose -f kafka-compose.yaml up -d
```

권한주기

```
chmod 777 configtxgen
```

채널생성

```
./configtxgen -profile AuctionOrdererGenesis -outputBlock orderer-genesis.block -channelID ordererchannel001
```

```
./configtxgen -profile AuctionChannel -outputCreateChannelTx auctionchannel001.tx -channelID auctionchannel001
```

생성된 트랜잭션 파일이동

```
mv auctionchannel001.tx ./channel-artifacts
```

Orderer 실행

```
sudo docker-compose -f orderer-compose.yaml up -d
```

Peer 실행

```
vi peer-compose.yaml 입력후
----> CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE 확인후
peer-compose.yaml이 속해있는 폴더이름_defalut로 변경 (반드시 소문자) -2군데있음

sudo docker-compose -f peer-compose.yaml up -d
```

채널생성

```
sudo docker exec -e "CORE_PEER_LOCALMSPID=copyrightAssociateOrgMSP" -e "CORE_PEER_MSPCONFIGPATH=/var/hyperledger/users/msp" peer0.copyrightOrg peer channel create -o orderer001:7050 -c auctionchannel001 -f /var/hyperledger/production/auctionchannel001.tx --outputBlock /var/hyperledger/production/auctionchannel001-genesis.block
```

채널 참여

```
sudo docker exec -e "CORE_PEER_LOCALMSPID=copyrightAssociateOrgMSP" -e "CORE_PEER_MSPCONFIGPATH=/var/hyperledger/users/msp" peer0.copyrightOrg peer channel fetch config /var/hyperledger/production/auctionchannel001-genesis.block -o orderer001:7050 -c auctionchannel001

sudo docker exec -e "CORE_PEER_LOCALMSPID=copyrightAssociateOrgMSP" -e "CORE_PEER_MSPCONFIGPATH=/var/hyperledger/users/msp" peer0.copyrightOrg peer channel join -o orderer001:7050 -b /var/hyperledger/production/auctionchannel001-genesis.block

sudo docker exec -e "CORE_PEER_LOCALMSPID=digitalAuthorOrgMSP" -e "CORE_PEER_MSPCONFIGPATH=/var/hyperledger/users/msp" peer0.digitalAuthorOrg peer channel fetch config /var/hyperledger/production/auctionchannel001-genesis.block -o orderer001:7050 -c auctionchannel001

sudo docker exec -e "CORE_PEER_LOCALMSPID=digitalAuthorOrgMSP" -e "CORE_PEER_MSPCONFIGPATH=/var/hyperledger/users/msp" peer0.digitalAuthorOrg peer channel join -o orderer001:7050 -b /var/hyperledger/production/auctionchannel001-genesis.block
```

create anchor peer update tx

앵커피어 업데이트 트랜잭션 생성

```
./configtxgen -profile AuctionChannel -outputAnchorPeersUpdate copyrightOrgAnchor.tx -channelID auctionchannel001 -asOrg copyrightAssociateOrgMSP

./configtxgen -profile AuctionChannel -outputAnchorPeersUpdate digitalAuthorOrgAnchor.tx -channelID auctionchannel001 -asOrg digitalAuthorOrgMSP
```


앵커피어 업데이트 트랜잭션 이동

```
mv copyrightOrgAnchor.tx ./channel-artifacts
mv digitalAuthorOrgAnchor.tx ./channel-artifacts
```

update anchor peer

앵커 피어 업데이트

```
sudo docker exec -e "CORE_PEER_LOCALMSPID=copyrightAssociateOrgMSP" -e "CORE_PEER_MSPCONFIGPATH=/var/hyperledger/users/msp"
peer0.copyrightOrg peer channel update -o orderer001:7050 -c auctionchannel001 -f /var/hyperledger/production/copyrightOrgAnchor.tx

sudo docker exec -e "CORE_PEER_LOCALMSPID=digitalAuthorOrgMSP" -e "CORE_PEER_MSPCONFIGPATH=/var/hyperledger/users/msp"
peer0.digitalAuthorOrg peer channel update -o orderer001:7050 -c auctionchannel001 -f
/var/hyperledger/production/digitalAuthorOrgAnchor.tx
```

체인코드

nodejs chaincode install

구동 확인을 위해 스켈레톤 체인코드가 아닌 완성본을 설치

```
sudo docker exec -e "CORE_PEER_LOCALMSPID=copyrightAssociateOrgMSP" -e "CORE_PEER_MSPCONFIGPATH=/var/hyperledger/users/msp"
peer0.copyrightOrg peer chaincode install -n asset -v 0.1 -l node -p /var/hyperledger/chaincode/node

sudo docker exec -e "CORE_PEER_LOCALMSPID=digitalAuthorOrgMSP" -e "CORE_PEER_MSPCONFIGPATH=/var/hyperledger/users/msp"
peer0.digitalAuthorOrg peer chaincode install -n asset -v 0.1 -l node -p /var/hyperledger/chaincode/node
```

nodejs chaincode instantiate

체인코드 도커 컨테이너를 생성 및 구동

```
sudo docker exec -e "CORE_PEER_LOCALMSPID=digitalAuthorOrgMSP" -e "CORE_PEER_MSPCONFIGPATH=/var/hyperledger/users/msp"
peer0.digitalAuthorOrg peer chaincode instantiate -o orderer001:7050 -C auctionchannel001 -n asset -v 0.1 -c '{"Args":["init"]}' -P
"OR('copyrightAssociateOrgMSP.member','digitalAuthorOrgMSP.member')"
```

nodejs chaincode invoke

생성된 체인코드 동작 테스트

```
sudo docker exec -e "CORE_PEER_LOCALMSPID=digitalAuthorOrgMSP" -e "CORE_PEER_MSPCONFIGPATH=/var/hyperledger/users/msp"
peer0.digitalAuthorOrg peer chaincode invoke -o orderer001:7050 -C auctionchannel001 -n asset -c '{"Args":
["registerAsset","testasset","testowner"]}'
```

nodejs chaincode query

생성된 체인코드에 쿼리를 날려 invoke 시 등록한 정보가 반환되면 정상동작

```
sudo docker exec -e "CORE_PEER_LOCALMSPID=digitalAuthorOrgMSP" -e "CORE_PEER_MSPCONFIGPATH=/var/hyperledger/users/msp"
peer0.digitalAuthorOrg peer chaincode query -o orderer001:7050 -C auctionchannel001 -n asset -c '{"Args":["query","testasset"]}'
```

하리보 백엔드에서 properties 수정후

//CA배포

```
sudo docker-compose -f ./certificate-authority/ca-compose.yaml up -d
```

10. 02 하이퍼패브릭 네트워크 및 체인코드 배포

노트북: 블록체인

만든 날짜: 2019-09-09 오전 11:12

업데이트:

2019-09-09 오전 11:17

작성자: 비니문

실행순서: configtxgen 파일생성 --> docker-compose 구동 및 확인 --> channel create --> channel join --> create anchor peer update tx --> update anchor peer --> chaincode install --> chaincode instantiate

configtxgen 파일생성

configtxgen --> 채널 구성이나, 업데이트되는(피어가 채널에 참여라던지, 연결정보 변경한다던지) 시에 필요한 정보 파일(~.tx 이나 ~.block 등) 생성

./configtxgen -profile AuctionOrdererGenesis -outputBlock orderer-genesis.block -channelID ordererchannel001

./configtxgen -profile AuctionChannel -outputCreateChannelTx auctionchannel001.tx -channelID auctionchannel001

생성된 파일을 이동시킴(auctionchannel001.tx만 이동하며, orderer-genesis.block은 이동시키지 않음)

이후에 노드파일이 사용함

mv auctionchannel001.tx ./channel-artifacts

4개의 도커 컴포즈 파일을 순서대로 구동시킨다(zookeeper-compose.yaml -> kafka-compose.yaml -> orderer-compose.yaml -> peer-compose.yaml)

zookeeper 실행

```
sudo docker-compose -f zookeeper-compose.yaml up -d
```

check zookeeper000

```
nc -z 0.0.0.0 2181
if [ "$?" != 0 ]; then
    echo 'zookeeper000 run fail. Please check docker log.'
else
    echo 'zookeeper000 is active.'
fi
```

check zookeeper001

```
nc -z 0.0.0.0 2182
if [ "$?" != 0 ]; then
    echo 'zookeeper001 run fail. Please check docker log.'
else
    echo 'zookeeper001 is active.'
fi
```

check zookeeper002

```
nc -z 0.0.0.0 2183
if [ "$?" != 0 ]; then
    echo 'zookeeper002 run fail. Please check docker log.'
else
    echo 'zookeeper002 is active.'
fi
```

kafka 실행 (kafka는 "docker logs kafka000" 명령어로 로그 확인 필요)

kafka 로그 상에서 [KafkaServer id=1] started 라는 로그 출력 시 정상 동작(id 값은 kafka별로 상이)

```
sudo docker-compose -f kafka-compose.yaml up -d
```

orderer 실행

```
sudo docker-compose -f orderer-compose.yaml up
```

check orderer001

```
nc -z 0.0.0.0 8050
if [ "$?" != 0 ]; then
    echo 'orderer001 run fail. Please check docker log.'
else
    echo 'orderer001 is active.'
fi
```

check orderer002

```
nc -z 0.0.0.0 7060
if [ "$?" != 0 ]; then
    echo 'orderer002 run fail. Please check docker log.'
else
    echo 'orderer002 is active.'
fi
```

check orderer003

```
nc -z 0.0.0.0 7070
if [ "$?" != 0 ]; then
    echo 'orderer003 run fail. Please check docker log.'
else
    echo 'orderer003 is active.'
fi
```

peer 실행

```
sudo docker-compose -f peer-compose.yaml up -d
```

check peer0.copyrightOrg

```
nc -z 0.0.0.0 7071
if [ "$?" != 0 ]; then
    echo 'peer0.copyrightOrg run fail. Please check docker log.'
else
    echo 'peer0.copyrightOrg is active.'
fi
```

check peer0.digitalAuthorOrg

```
nc -z 0.0.0.0 8051
if [ "$?" != 0 ]; then
    echo 'peer0.digitalAuthorOrg run fail. Please check docker log.'
else
    echo 'peer0.digitalAuthorOrg is active.'
fi
```

channel create

피어들끼리 통신할 채널 생성

peer channel create -o orderer001:7050 -c auctionchannel001 -f /var/hyperledger/fabric/production/auctionchannel001.tx --outputBlock

```
/var/hyperledger/production/auctionchannel001-genesis.block
```

```
docker exec -e "CORE_PEER_LOCALMSPID=copyrightAssociateOrgMSP" -e "CORE_PEER_MSPCONFIGPATH=/var/hyperledger/users/msp"
peer0.copyrightOrg peer channel create -o orderer001:7050 -c auctionchannel001 -f /var/hyperledger/production/auctionchannel001.tx --
outputBlock /var/hyperledger/production/auctionchannel001-genesis.block
```

channel join

```
docker exec -e "CORE_PEER_LOCALMSPID=copyrightAssociateOrgMSP" -e "CORE_PEER_MSPCONFIGPATH=/var/hyperledger/users/msp"
peer0.copyrightOrg peer channel fetch config /var/hyperledger/production/auctionchannel001-genesis.block -o orderer001:7050 -c
auctionchannel001

docker exec -e "CORE_PEER_LOCALMSPID=copyrightAssociateOrgMSP" -e "CORE_PEER_MSPCONFIGPATH=/var/hyperledger/users/msp"
peer0.copyrightOrg peer channel join -o orderer001:7050 -b /var/hyperledger/production/auctionchannel001-genesis.block

docker exec -e "CORE_PEER_LOCALMSPID=digitalAuthorOrgMSP" -e "CORE_PEER_MSPCONFIGPATH=/var/hyperledger/users/msp"
peer0.digitalAuthorOrg peer channel fetch config /var/hyperledger/production/auctionchannel001-genesis.block -o orderer001:7050 -c
auctionchannel001

docker exec -e "CORE_PEER_LOCALMSPID=digitalAuthorOrgMSP" -e "CORE_PEER_MSPCONFIGPATH=/var/hyperledger/users/msp"
peer0.digitalAuthorOrg peer channel join -o orderer001:7050 -b /var/hyperledger/production/auctionchannel001-genesis.block
```

create anchor peer update tx

```
./configtxgen -profile AuctionChannel -outputAnchorPeersUpdate copyrightOrgAnchor.tx -channelID auctionchannel001 -asOrg
copyrightAssociateOrgMSP

./configtxgen -profile AuctionChannel -outputAnchorPeersUpdate digitalAuthorOrgAnchor.tx -channelID auctionchannel001 -asOrg
digitalAuthorOrgMSP

mv copyrightOrgAnchor.tx ./channel-artifacts

mv digitalAuthorOrgAnchor.tx ./channel-artifacts
```

update anchor peer

```
docker exec -e "CORE_PEER_LOCALMSPID=copyrightAssociateOrgMSP" -e "CORE_PEER_MSPCONFIGPATH=/var/hyperledger/users/msp"
peer0.copyrightOrg peer channel update -o orderer001:7050 -c auctionchannel001 -f /var/hyperledger/production/copyrightOrgAnchor.tx

docker exec -e "CORE_PEER_LOCALMSPID=digitalAuthorOrgMSP" -e "CORE_PEER_MSPCONFIGPATH=/var/hyperledger/users/msp"
peer0.digitalAuthorOrg peer channel update -o orderer001:7050 -c auctionchannel001 -f
/var/hyperledger/production/digitalAuthorOrgAnchor.tx
```

체인코드

nodejs chaincode install

구동 확인을 위해 스크레톤 체인코드가 아닌 완성본을 설치

```
docker exec -e "CORE_PEER_LOCALMSPID=copyrightAssociateOrgMSP" -e "CORE_PEER_MSPCONFIGPATH=/var/hyperledger/users/msp"
peer0.copyrightOrg peer chaincode install -n asset -v 0.1 -l node -p /var/hyperledger/chaincode/node

docker exec -e "CORE_PEER_LOCALMSPID=digitalAuthorOrgMSP" -e "CORE_PEER_MSPCONFIGPATH=/var/hyperledger/users/msp"
peer0.digitalAuthorOrg peer chaincode install -n asset -v 0.1 -l node -p /var/hyperledger/chaincode/node
```

nodejs chaincode instantiate

체인코드 도커 컨테이너를 생성 및 구동

```
docker exec -e "CORE_PEER_LOCALMSPID=digitalAuthorOrgMSP" -e "CORE_PEER_MSPCONFIGPATH=/var/hyperledger/users/msp"
peer0.digitalAuthorOrg peer chaincode instantiate -o orderer001:7050 -C auctionchannel001 -n asset -v 0.1 -c '{"Args":["init"]}' -P
"OR('copyrightAssociateOrgMSP.member', 'digitalAuthorOrgMSP.member')"
```

nodejs chaincode invoke

생성된 체인코드 동작 테스트

```
docker exec -e "CORE_PEER_LOCALMSPID=digitalAuthorOrgMSP" -e "CORE_PEER_MSPCONFIGPATH=/var/hyperledger/users/msp"
peer0.digitalAuthorOrg peer chaincode invoke -o orderer001:7050 -C auctionchannel001 -n asset -c '{"Args":
["registerAsset", "testasset", "testowner"]}'
```

nodejs chaincode query

생성된 체인코드에 쿼리를 날려 invoke 시 등록된 정보가 반환되면 정상동작

```
docker exec -e "CORE_PEER_LOCALMSPID=digitalAuthorOrgMSP" -e "CORE_PEER_MSPCONFIGPATH=/var/hyperledger/users/msp"
peer0.digitalAuthorOrg peer chaincode query -o orderer001:7050 -C auctionchannel001 -n asset -c '{"Args":["query", "0"]}'
```

nodejs chaincode package

체인코드 수정 후 재배포를 위한 패키지 생성 명령어

-v 플래그에 버전을 기록

```
docker exec -e "CORE_PEER_LOCALMSPID=digitalAuthorOrgMSP" -e "CORE_PEER_MSPCONFIGPATH=/var/hyperledger/users/msp"
peer0.digitalAuthorOrg peer chaincode package /var/hyperledger/chaincode/ccpack.out -n asset -v 0.2 -l node -p
/var/hyperledger/chaincode/node
```

nodejs chaincode package install

```
docker exec -e "CORE_PEER_LOCALMSPID=copyrightAssociateOrgMSP" -e "CORE_PEER_MSPCONFIGPATH=/var/hyperledger/users/msp"
peer0.copyrightOrg peer chaincode install /var/hyperledger/chaincode/ccpack.out -p /var/hyperledger/chaincode/node

docker exec -e "CORE_PEER_LOCALMSPID=digitalAuthorOrgMSP" -e "CORE_PEER_MSPCONFIGPATH=/var/hyperledger/users/msp"
peer0.digitalAuthorOrg peer chaincode install /var/hyperledger/chaincode/ccpack.out -p /var/hyperledger/chaincode/node
```

nodejs chaincode upgrade

체인코드 수정 후 설치된 버전으로 업그레이드를 수행하는 명령어

```
docker exec -e "CORE_PEER_LOCALMSPID=digitalAuthorOrgMSP" -e "CORE_PEER_MSPCONFIGPATH=/var/hyperledger/users/msp"
peer0.digitalAuthorOrg peer chaincode upgrade -o orderer001:7050 -C auctionchannel001 -n asset -v 0.4 -c '{"Args":["init"]}' -P
"OR('copyrightAssociateOrgMSP.member','digitalAuthorOrgMSP.member')"
```

10. 03 체인코드 소스

노트북:	블록체인	업데이트:	2019-09-09 오전 11:13
만든 날짜:	2019-09-03 오전 11:04		
작성자:	비니문		
URL:	https://fabric-shim.github.io/master/fabric-shim.ChaincodeStub.html#createCompositeKey__anchor		

Init

```
async Init(stub){
  console.info('Instantiated completed');
  return shim.success();
}
```

Invoke

```
async Invoke(stub){

  /** 체인 코드 인수에서 메소드 이름 및 매개 변수 가져 오기 */
  let ret = stub.getFunctionAndParameters();
  let method = this[ret.fcn];

  /** 정의되지 않은 메소드 호출 예외 (오류는 발생하지 않음) */
  if(!method){
    console.log('Method name [' + ret.fcn + '] is not defined');
  }

  /** Method call */
  try{
    let payload = await method(stub, ret.params);
    return shim.success(payload);
  }catch(err){
    console.log(err);
    return shim.error(err);
  }
}
```

호출 할 체인 코드 함수 이름과 대상 함수에 전달할 인수 배열을 포함하는 객체를 반환합니다.

registerAsset

```
async registerAsset(stub, args){

  /** 부적절한 인수 예외 */
  if(args.length != 2){
    throw new Error('Incorrect number of arguments. Expecting 2, but received '+ args.length);
  }

  /** !!! 복합 키 생성 !!! */
  let compositeKey = stub.createCompositeKey("Asset.", [args[0]]);

  /** 중복 자산 검사 */
  let dupCheck = await stub.getState(compositeKey);

  var isExist = function(value){
    if(value == "" || value == null || value == undefined ||
      (value != null && typeof value == "object" && !Object.keys(value).length)){
      return true;
    }

    else{
      return false;
    }
  };

  if(isExist(dupCheck) != true){
    throw new Error('AssetID ' + compositeKey + 'is already registered.');
```

confirmTimestamp__자산 등록 시간 인 타임 스탬프를 확인하십시오.

```
async confirmTimestamp(stub, args){
  /** 부적절한 인수 예외 */
  if(args.length != 1){
    throw new Error('Incorrect number of arguments. Expecting assetID as an argument');
  }

  /** !!! 복합 키 생성 !!! */
  let searchKey = stub.createCompositeKey("Asset.", [args[0]]);

  /** 자산 정보 상태 열기 */
  let asset = await stub.getState(searchKey);
  let assetInfo = JSON.parse(asset);

  /** 'stub'를 사용하여 거래 타임 스탬프 가져 오기 */
  let txTimestamp = stub.getTxTimestamp();

  /** 타임 스탬프 형식 'YYYY-MM-DD HH:MM:SS' */
```

```

    let timestampString;
    let tsSec = txTimestamp.seconds;
    let tsSecValue = tsSec.low;
    let dateTimeObj = new Date(tsSecValue*1000);

    timestampString = dateTimeObj.getFullYear() + '-' + ('0' + (dateTimeObj.getMonth() + 1)).slice(-2) + '-' +
    ('0'+dateTimeObj.getDate()).slice(-2)+' ' + (dateTimeObj.getHours() + 9) + ':'+'0'+dateTimeObj.getMinutes()).slice(-2)+
    ':'+dateTimeObj.getSeconds();

    /** 자산의 createdAt 필드 수정 */
    assetInfo.createdAt = timestampString;

    /** 수정 된 자산 정보를 넣습니다 */
    await stub.putState(searchKey, Buffer.from(JSON.stringify(assetInfo)));

}

```

query__디지털 자산 정보를 얻습니다.

```

*/
async query(stub, args){

    /** !!! Generate composite key !!! */
    let searchKey = stub.createCompositeKey("Asset.", [args[0]]);

    /** Get state */
    let asset = await stub.getState(searchKey);

    /** Return asset state */
    return asset;
}

```

11. Hyperledger Fabric SDK Java SDK

노트북: 블록체인

만든 날짜: 2019-09-05 오전 10:40

업데이트:

2019-09-05 오전 10:42

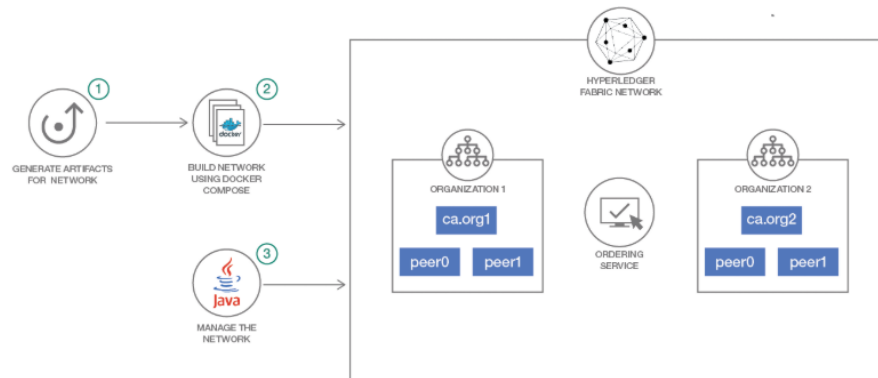
작성자: 비니문

URL: <https://developer.ibm.com/patterns/create-and-deploy-blockchain-network-using-fabric-sdk-java/>

Hyperledger Fabric SDK Java SDK

: 채널에서 사용자 체인 코드, 쿼리 블록 및 트랜잭션을 실행하고 해당 채널의 이벤트를 모니터링하는 수단을 제공

흐름



1. 네트워크의 피어 및 채널에 대해 crypttx 및 configtx를 사용하여 아티팩트를 생성하십시오. 이들은 이미 생성되어 그대로 코드 저장소에서 제공됩니다.
2. Docker Compose와 생성된 아티팩트를 사용하여 네트워크를 빌드하십시오.
3. Hyperledger Fabric Java SDK API를 사용하여 네트워크 작업 및 관리:
 1. 채널을 작성하고 초기화하십시오.
 2. 체인 코드를 설치하고 인스턴스화하십시오.
 3. 호출 및 조회를 수행하여 네트워크를 테스트하십시오.

11. 페브릭CC서비스

노트북: 블록체인

만든 날짜: 2019-09-04 오전 10:56

업데이트: 2019-09-17 오후 3:30

작성자: 비니문

getPropertiesWith

```
private Properties getPropertiesWith(String filename) {
    Properties properties = new Properties();
    properties.put("pemBytes", CommonUtil.readString(filename).getBytes());
    properties.setProperty("sslProvider", "openSSL");
    properties.setProperty("negotiationType", "TLS");
    return properties;
}
```

registerOwnership

```
* 소유권 등록을 위해 체인코드 함수를 차례대로 호출한다.
* @param 소유자
* @param 작품id
* @return FabricAsset

public FabricAsset registerOwnership(final long 소유자, final long 작품id){
    if(this.channel == null)
        loadChannel();
    boolean res = registerAsset(작품id, 소유자);
    if(!res)
        return null;
    res = confirmTimestamp(작품id);
    if(!res)
        return null;
    return query(작품id);
}
```

transferOwnership

```
* 소유권 이전을 위해 체인코드 함수를 차례대로 호출한다.
* @param from
* @param to
* @param 작품id
* @return List<FabricAsset>
@Override
public List<FabricAsset> transferOwnership(final long from, final long to, final long 작품id) {
    if(this.channel == null)
        loadChannel();
    List<FabricAsset> assets = new ArrayList<>();
    boolean res = this.expireAssetOwnership(작품id, from);
    if(!res) return null;
    FabricAsset expired = query(작품id);
    if(expired == null) return null;
    assets.add(expired);
    res = this.updateAssetOwnership(작품id, to);
    if(!res) return null;
    FabricAsset transferred = query(작품id);
    if(transferred == null) return null;
    assets.add(transferred);
    return assets;
}
```


12. 이더리움 스마트 컨트랙트 배포

노트북: 블록체인

만든 날짜: 2019-09-17 오후 3:30

업데이트: 2019-09-17 오후 9:56

작성자: 비니문

URL: https://winterj.me/prepare_smart_contract_deploying/

배포 흐름

1. solidity로 스마트 컨트랙트 작성
2. web3 혹은 solc를 이용해 바이트코드로 컴파일
3. 컴파일의 결과물인 바이트코드과 ABI를 얻음
4. testrpc 실행
5. web3를 이용해 localhost:port에 있는 testrpc에 바이트코드를 배포. 이 때 ABI가 사용됨
6. 배포 후 컨트랙트 어드레스를 반환받을 수 있음
7. web3를 이용해 컨트랙스 어드레스와 ABI를 이용해 컨트랙트를 가져와 함수를 실행시킬 수 있음
8. web3.js와 html를 연동시켜 웹 페이지와 블록체인에 있는 스마트 컨트랙트가 서로 통신할 수 있음

13. 하이퍼 레저 패브릭 공부

노트북: 블록체인

만든 날짜: 2019-09-24 오후 4:57

업데이트: 2019-09-24 오후 4:57

작성자: 비니문

<https://medium.com/hexlant/%EC%9D%B4%EB%8D%94%EB%A6%AC%EC%9B%80-keystore-%ED%8C%8C%EC%9D%BC-utc-%EC%83%9D%EC%84%B1-%EB%B0%8F-%EC%95%94%ED%98%B8%ED%99%94-%EB%B3%B5%ED%98%B8%ED%99%94-%EC%9B%90%EB%A6%AC-1-2-d417cb605bf>

14. 블록체인 기반 저작물 거래 플랫폼 서비스

노트북:	블록체인	업데이트:	2019-10-10 오후 2:06
만든 날짜:	2019-10-10 오전 11:34		
작성자:	비니문		

*문제-해결점-수단-돈

경매 시스템과 저작권 침해(특히 이미지)의 문제점을 이야기로 시작

그리고 이를 해결하기 위해서 블록체인을 선택했다

<블록체인이 왜 안전한지, 하이퍼레저와 이더리움의 정의와 이를 어떻게 엮어서 경매 사이트를 만드는지>

어떤한 프로그램 써서 개발하는지 설명하고

기능설명

이를 향후 발전 가능성

왜 블록체인 기반의 경매 사이트인가?

블록체인 기술은 분산화·암호화·확장성·데이터의 투명성이 특징으로 P2P 형태로 정보를 공유하면서 보호하는 것을 목적으로 하여 최근 저작권 분야에서 이익배분의 문제를 해결하기 위한 대안으로 주목 받고 있음

블록체인의 흐름

비즈니스 네트워크의

- ① 참여자(participants)들이 가지고 있는
- ② 유무형의 자산(assets)*을
- ③ 스마트 계약(smart contract)**을 기반으로
- ④ 거래(transaction)를 투명하게 공유하는 기술

>>>모든 구성원이 분산형 네트워크를 통해 정보 및 가치를 검증·저장·실행함으로써 특정인의 임의적인 조작이 어렵도록 설계된 분산 시스템 기술

>>>공유원장(ledger), 스마트 계약, 프라이버시 및 보안, 합의(참여자 동의)
→ 시간절약, 비용절감, 위험감소, 신뢰확산

표 1. 블록체인의 종류	
구분	개념 및 특징
퍼블릭 블록체인 (Public Blockchain)	<ul style="list-style-type: none">최초의 블록체인 활용사례인터넷을 통해 모두에게 공개, 운용 가능한 거래장부누구든지 검증 및 거래 참여가능네트워크의 확장이 어렵고 거래속도가 느림
프라이빗 블록체인 (Private Blockchain)	<ul style="list-style-type: none">원장을 관리할 수 있는 사람들 사전에 스크린(pre-screen)하나의 주체가 내부 자산만을 블록체인으로 관리Private Blockchain 개발을 위한 플랫폼 서비스도 등장
하이브리드 블록체인 (Hybrid Blockchain)	<ul style="list-style-type: none">반중앙형 블록체인미리 선정된 노드에 의해 조정되고 n개의 주체들이 노드를 1개씩 운영n개의 주체들 간에 합의된 룰에 따라 검증 참여네트워크 확장 용이하고 거래속도 빠름

지식재산 패러다임의 변화 필요 → 보호대상의 확대, 이용의 활성화 중요

블록체인의 탈중앙화, P2P 거래방식을 이용하여 음악, 사진, 영상 등 콘텐츠 창작자가 직접 소비자에게 제공하고, 소비자가 직접 창작자에게 보상하는 구조

슬라이드1

15. 블록체인 최종 정리

노트북: 블록체인

만든 날짜: 2019-10-20 오후 6:26

업데이트: 2019-10-20 오후 6:30

작성자: 비니문

문(제)해(결점)수(단)돈

문제

- 저작권 문제의 침해와 활성화
 - 이미지 저작권의 침해
 - 이미지 저작권 활성화 방안
 - 이미지 저작권 판매
 - 사진작가가 아닌 일반인들도 접근가능한 루트

해결점

- 이미지 저작권 경매 사이트
 - 경매 사이트를 통하여 사진작가는 물론 일반인들도 접근 가능하게

수단

- 일반 이미지 저작권 경매 사이트는 보안에 취약함
 - 블록체인 기반으로 기획
 - 프라이빗 블록체인(하이퍼 레저 패브릭)+퍼블릭 블록체인(이더리움)

돈

- 경매 수수료
- 사이트 배너 광고
- 코인 충전 수수료