# Chapter 13. NFT

NFT(Nun-Fungible Token)는 ERC721의 표준을 지켜서 구현해야 한다. NFT는 디지털 저작물을 블록체인에 저장하고, 소유권을 지정하는 것이다. 제작된 NFT를 시장에서 판매하는 방법을 설명한다.

# 1. NFT가 대체 무엇일까?

NFT은 디지털 자산이다. 사진, 그림, 게임의 아이템, 비디오, 부동산, 소프트웨어 라이센스, 입장권 어떤 것이나 NFT로 만들어질 수 있다.

디지털 자산은 음악이나 사진이나 온라인에 널려 있어 새삼스러운 것도 아니다. 그렇다면 기존의 그런 것들과는 어떤 차이가 있길래 유난을 떠는 것일까? 디지털로 만들었다면 모두 NFT일까? 그렇지 않다, 블록체인 상에 존재하며 그래서, NFT는 어떤 저작물의 소유를 증명하기 때문에 누구도 소유권을 위조할 수 없다.

DeFi는 교환이 가능한 화폐를 대상으로 하고 있고, NFT는 디지털 자산이 될 수 있는 미술품, 부동산, 복권, 주식 어떤 것이든 대상이 될 수 있다는 점에 매력이 있다.

NFT는 말 그대로 대체 불가능을 의미하는 Non Fungible이다. DeFi는 대체 가능해서(Fungible) 대조적인데, Fungible은 어떻길래 이런 단어를 사용하는 것일까? 상호교환이 불가능하다는 것은 NFT 상호간에 교환이 될 수 없다는 것이다. 반면에 DeFi는 상호 교환, 즉 비트코인이나 다른 DeFi와 서로 교환이 가능하다는 것이다.

최초의 NFT는 2014년 5월 Kevin McCoy가 만든 "Quantum"이다. 8각형을 여러 디지털 이미지로 변화해서 보여주고 있다. 그러다가 NFT가 보다 널리 알려지게 된 것은 2017년 블록체인 게임 '크립토키티'이다. 최근에는 수천장의 사진을 콜라쥬한 작품 "Everydays: the First 5000 Days"이 NFT로 만들어져서 고가에 거래돼 화제를 모았다. 트위터의 창업자인 Jack Dorsey의 첫 트윗도 NFT로 만들어지는 등 NFT가 인기를 끌고 있다.

# 2. 제작하는 방법

## 2.1 스마트 컨트랙을 제작

PNG 이미지가 있고 이를 NFT로 만든다고 하자. 무엇을 해야 할까?

간단하다. 그 이미지 자체와 소유권이 누구에게 있는지 정해서 블록체인에 저장하면 된다. 정확히 말하면, 이미지, 이미지의 메타데이터 및 소유권을 증명하는 코딩을 해야 하고, 앞서 했던 것처럼 컨트랙을 만들어서 블록체인에 배포를 하는 것이다.

## 2.2 이미지는 블록체인에 저장하지 않는다

이미지는 (1) NFT 컨트랙에 저장하거나 (이 방법이 직관적이기는 하다) 또는 (2) 분산 공유 파일 시스템인 IPFS(Inter Planetary File System)에 저장할 수 있다.

이미지를 컨트랙에 저장하면 좋지만, 그렇게 하지 않는 편이 좋다. 왜 저장하지 않을까? 물론 할 수 있다. 이미지 하나라면 용량이 얼마되지 않겠지만 수 백개, 수 천개라고 하면 비용이 만만치 않다. 스마트 컨트랙은 큰 파일, 이미지, 동영상을 저장하는 용도로는 비용적인 측면에서 바람직하지 않다. 이미지가 작더라도 컨트랙에 저장한다면 결국 gas 비용이 소요되기 때문에 권장하지 않는다.

이더리움 황서(yellow paper)에 따르면, 256 비트를 저장하려면 20k의 gas가 소요된다. 이미지 자체를 저장하기 위해 수 MB 또는 그 이상의 용량을 저장한다면 gas 비용이 급격히 증가할 것이다. 메타데이터인 JSON 내용을 저장한다면 역시 비용이 수반된다. 이러한 비용을 회피하기 위해 컨트랙에는 이미지나 메타데이터들을 저장하는 대신 URL만 저장하는 방법이 현명하다고 하겠다.

NFT는 tokenURI가 있어야 한다. 디지털 자산에 대한 JSON 메타데이터로서, 명칭, 이미지, 설명 등을 저장한다. 예를 들면:

{  
 "attributes": [  
 {  
 "trait\_type": "Mood", 속성의 종류를 적는다  
 "value": "Happy" 위 속성의 값을 적는다  
 },  
 ... 여러 속성을 정의할 수 있다.  
 ],  
 "description": "A Happy Face", 설명을 적는다. 마크다운 가능.  
 "image": "https://example.com/HappyFace.png", 이미지 url을 적는다  
 "name": "Happy Face" 항목의 명칭을 적는다  
}

# 3. IERC721

NFT를 만들려면 ERC721 표준을 따라야 한다. IERC721는 인터페이스로서 ERC721 구현에 필요한 함수와 이벤트를 가지고 있다. ERC721 코드는 IERC721을 상속하여 코딩하면 쉽게 안전한 코드를 작성할 수 있다. 여기서는 함수명만 적고, 설명은 인터페이스를 구현한 ERC721에서 곧 이어진다.

## 3.1 소유

* function balanceOf(address owner) external view returns (uint256 balance)
* function ownerOf(uint256 tokenId) external view returns (address owner)

## 3.2 이체

* event Transfer(address indexed from, address indexed to, uint256 indexed tokenId)
* function safeTransferFrom(address from, address to, uint256 tokenId) external
* function safeTransferFrom(address from, address to, uint256 tokenId, bytes calldata data) external
* function transferFrom(address from,address to,uint256 tokenId) external

## 3.3 허가

* event Approval(address indexed owner, address indexed approved, uint256 indexed tokenId)
* event ApprovalForAll(address indexed owner, address indexed operator, bool approved)
* function approve(address to, uint256 tokenId) external;int256 tokenId) external
* function getApproved(uint256 tokenId) external view returns (address operator)
* function setApprovalForAll(address operator, bool \_approved) external
* function isApprovedForAll(address owner, address operator) external view returns (bool)

# 4. ERC721

NFT 관련 표준 EIP-721에서는 ERC721 토큰을 생성할 경우 따라야 하는 기본 내용을 정하고 있다. 물론 이더와 같은 화폐도 토큰으로 볼 수 있지만, 해당이 없다. 여기서는 사용자가 직접 만들어서 토큰의 명칭을 붙일 수 있는 그런 Non-Fungible 토큰을 말한다.

객체 지향의 상속을 활용하고 있는데, 아래와 같은 상위 컨트랙 또는 인터페이스를 구현하고 있다.

* IERC721 - 앞서 설명하고 있다.
* Context - 위 ERC20에서 사용하는 Context와 동일하게 msgSender, msgData를 정의한다.
* ERC165 - supportsInterface(bytes4 interfaceId)
* IERC721Metadata - name, symbol, tokenURI를 선언한다.

## 4.1 매핑

* mapping(uint256 => address) private \_owners token ID의 소유자 주소
* mapping(address => uint256) private \_balances 주소가 소유한 토큰 개수
* mapping(uint256 => address) private \_tokenApprovals token ID의 승인된주소
* mapping(address => mapping(address => bool)) private \_operatorApprovals Mapping from owner to operator approvals

## 4.2 생성자

* constructor(string memory name\_, string memory symbol\_) 명칭과 심볼을 생성

## 4.3 명칭 및 심볼

앞서 생성자에서 정의된 명칭과 심볼을 출력하는 함수이다.

* function name() public view virtual override returns (string memory) token의 명칭을 출력하는 함수이다. virtual함수라서 재정의할 수 있다.
* function symbol() public view virtual override returns (string memory) token의 symbol을 반환하는 함수이고, virtual함수라서 재정의할 수 있다.

## 4.4 소유 및 URI

몇 개의 NFT를 소유하고 있는지 확인하는 함수 balanceOf()와 NFT 일련번호의 소유주를 알아보는 ownerOf() 함수이다.

* function balanceOf(address owner) public view virtual override returns (uint256) IERC721-balanceOf를 재정의한 함수이며, 동시에 virtual로 자식이 재정의하는 것이 가능하다. \_balances[owner], owner가 소유한 NFT의 개수를 반환한다. 잔고로 착각하기 쉽다.
* function ownerOf(uint256 tokenId) public view virtual override returns (address) IERC721-ownerOf를 재정의한 함수이며, 동시에 virtual로 자식이 재정의할 수 있다. tokenId의 NFT를 소유한 owner 주소를 반환한다.
* function tokenURI(uint256 tokenId) public view virtual override returns (string memory) \_owners[tokenId], toeknId의 NFT가 저장된 URI를 반환

## 4.5 허가

소유자가 아닌 경우 이체를 하려면 소유권을 부여받고 나서야 가능하다.

* 단계 1: approve, 소유권을 이전하기 위해서는 권한을 부여받고, 그리고 나서
* 단계 2: transferFrom 소유권을 이전한다.

다음은 허가와 관련된 함수들이다.

* function approve(address to, uint256 tokenId) public virtual override IERC721의 approve 함수를 재정의하고, tokenId의 owner가 token을 to에게 양도 허용한다. 내부 함수 \_approve(to, tokenId)를 호출하면, \_tokenApprovals[tokenId] = to로 변경한다.
* function getApproved(uint256 tokenId) public view virtual override returns (address) tokenId의 owner주소 \_tokenApprovals[tokenId]를 반환한다.
* function setApprovalForAll(address operator, bool approved) public virtual override
* function isApprovedForAll(address owner, address operator) public view virtual override returns (bool)

## 4.6 이체

소유권 이전과 관련한 함수들이다.

* function transferFrom(address \_from, address \_to, uint256 \_tokenId) external payable 앞의 ERC20에도 있는 함수이다. NFT 소유권을 이전한다. **함수 호출자가 from이 아니면, 허가를 받고 나서** 해야 한다 (approve 또는 setApprovalForAll). 토큰 소유자 또는 허가된 사용자가 아니면 오류가 발생한다.
* function safeTransferFrom(address from, address to, uint256 tokenId) public virtual override safeTransferFrom()은 to가 받을 수 있는지 확인하고 transfer하게 된다. 아래 코드에서 보듯이, 받는 측이 컨트랙일 경우 IERC721Receiver를 구현해서 받을 수 있게 하고, 그래야 NFT를 보냈을 때 전달되지 못하고 유실되지 않도록 안전하게 이전된다.

if (to.isContract())   
 IERC721Receiver(to).onERC721Received(\_msgSender(), from, tokenId, data) returns (bytes4 retval) {  
 return retval == IERC721Receiver.onERC721Received.selector;  
 }  
else return true

* function safeTransferFrom(address from, address to, uint256 tokenId, bytes data) public virtual override 위와 동일하지만 전달할 추가 데이터가 았는 경우에 사용한다

## 4.7 내부 함수들

컨트랙 내부에서만 호출될 수 있는 함수들로 공통적으로 \_로 시작하고 있다.

* function \_baseURI() internal view virtual returns (string memory) token URI를 설정
* function \_transfer(address from, address to, uint256 tokenId) internal virtual tokenId NFT를 from에서 to로 소유권을 양도. from은 owner이거나 approved된 경우에만 가능
* function \_safeTransfer(address from,address to,uint256 tokenId,bytes memory \_data) internal virtual
* function \_exists(uint256 tokenId) internal view virtual returns (bool)
* function \_isApprovedOrOwner(address spender, uint256 tokenId) internal view virtual returns (bool)
* function \_safeMint(address to, uint256 tokenId) internal virtual
* function \_safeMint(address to, uint256 tokenId, bytes memory \_data) internal virtual
* function \_mint(address to, uint256 tokenId) internal virtual \_mint()는 과금이 있을 경우 사용한다. \_safemint()는 임의의 금액을 과금하는 함수라서 적합하지 않다. function \_burn(uint256 tokenId) internal virtual
* function \_approve(address to, uint256 tokenId) internal virtual
* function \_setApprovalForAll(address owner, address operator, bool approved) internal virtual
* function \_beforeTokenTransfer(address from, address to, uint256 tokenId) internal virtual
* function \_afterTokenTransfer(address from, address to, uint256 tokenId) internal virtual
* function \_checkOnERC721Received(address from, address to, uint256 tokenId, bytes memory \_data) private returns (bool)

# 5. IPFS

이미지를 인터넷에 업로드하는 것도 생각해 볼 수 있다. 컨트랙에 이미지 등을 저장하는 것은 비용 문제때문에 효율적이지 않다고 했다. 그럼 Dropbox, OneDrive, Google Drive, 또는 Github 같은 클라우드 서비스에 올리는 것은 어떨까? 이런 저장소에 저장하는 것은 이미지를 위변조할 수 있다는 단점이 있다. 경우에 따라서는 서비스가 중지되면 이미지 URL도 작동하지 않게 된다.

이러한 이유로 분산 저장소인 IPFS(InterPlanetary File System)를 사용하는 것을 고려할 수 있다. 이런 파일은 블록으로 쪼개져서, 해시값 CID(Content Identifier)를 가지게 된다. 이 블록들은 분산된 피어(peer)들에게 저장되고, 머클트리로 연결된다. IPFS는 CID가 해시값이므로 기억하기 어렵기 마련이지만, 문자열로 기억할 수 있게 명칭을 만들어준다.

기존 클라우드 저장 방법과 비교하면, 기존 방법은 파일을 업로드 한 후 URL이 주어진다. 그 URL을 이용해서 접속하면 파일에 접근할 수 있다. IPFS는 내용을 해싱하고 그 해시 값이 URL에 해당한다.

파일을 가져올 때는, 해시가 어디 있는지 찾는다. 그 해시에 해당하는 내용이 저장된 피어에 접속해서, 블록들을 불러오고, 연결하고 머클트리가 변경되지 않았는지 검증하게 된다.

IPFS에 NFT와 메타데이터를 저장하여 보자.

## 5.1 이미지 올리기

Pinata(https://www.pinata.cloud)는 1G 용량까지 무료로 제공하는 IPFS 서비스이다. Pinata에 파일을 직접 올리거나, API를 사용해서 올릴 수 있다. 이미지는 직접 올리면 결과 이미지의 해시를 획득할 수 있다. 예를 들어 획득한 해시가 'QmZSMw1PARg1dDxaN2CoNbskraRXJo7trUsoUickrRgrTn'라고 가정하자. 해시 값을 url에 붙여주면: https://ipfs.io/ipfs/QmZSMw1PARg1dDxaN2CoNbskraRXJo7trUsoUickrRgrTn 업로드한 파일에 접근할 수 있다.

웹브라우저에 이 페이지를 로딩해보자. 시간이 어느 정도 소요가 된다.

## 5.2 TokenURI 올리기

이번에는 API를 사용해서 올려보자. URI 파일을 작성해서 올려보자.

[파일명: src/nftHelloUri.json]  
{  
 "name": "Hello DApp Screens",  
 "description": "Screens to Deploy the Hello DApp",  
 "image": "https://ipfs.io/ipfs/QmZSMw1PARg1dDxaN2CoNbskraRXJo7trUsoUickrRgrTn"  
}

## 5.3 IPFS 서비스 제공자의 API를 활용

먼저 라이브러리를 설치한다.

pip install pinatapy-vourhey

그리고 Pinata의 API key, secret를 생성해 놓아야 한다.

* API Key: 33a50cd294...7ce76ec
* API Secret: 1f3be99b4a4167ad255df835d6b5303...cddb4e9d9f3a0fee6b0615529fee45

그리고 인증 키를 사용해 서비스에 연결한다.

>>> from pinatapy import PinataPy  
>>> APIKey="33a50cd294...7ce76ec"  
>>> APISecret="1f3be99b4a4167ad255df835d6b5303...cddb4e9d9f3a0fee6b0615529fee45"  
>>> pinata = PinataPy(APIKey, APISecret)

파일을 다시 업로드해도 해시는 동일하다.

>>> # Upload the file  
>>> result = pinata.pin\_file\_to\_ipfs("src/nftHelloUri.json")  
>>> # Should return the CID (unique identifier) of the file  
>>> print(result)  
  
{'IpfsHash': 'QmbdwcmzYacLZHLBAhAsaJm2aPV1bzBjFjweD1UgZzkEtA', 'PinSize': 192, 'Timestamp': '2022-04-14T07:51:47.038Z', 'isDuplicate': True}

대기 중인 업로드 작업이 있는지 출력한다.

>>> # Anything waiting to be done?  
>>> print(pinata.pin\_jobs())  
  
{'count': 0, 'rows': []}

>>> # Total data in use  
>>> print(pinata.user\_pinned\_data\_total())  
  
{'pin\_count': 2, 'pin\_size\_total': '148583', 'pin\_size\_with\_replications\_total': '148583'}

업로드한 파일이 몇 개인지 알아본다. 'ipfs\_pin\_hash' 해시가 파일명이 된다. 즉 https://ipfs.io/ipfs/해시라고 url이 완성된다.

>>> # List of items we have pinned so far  
>>> print(pinata.pin\_list())  
  
{'count': 2, 'rows': [{'id': '07c804c5-66cf-4b79-8c63-218dad6869dc', 'ipfs\_pin\_hash': 'QmbdwcmzYacLZHLBAhAsaJm2aPV1bzBjFjweD1UgZzkEtA', 'size': 192, 'user\_id': '7be18edd-36fd-41c9-80ad-7a950eb7e104', 'date\_pinned': '2022-04-14T07:51:47.038Z', 'date\_unpinned': None, 'metadata': {'name': 'nftHelloUri.json', 'keyvalues': None}, 'regions': [{'regionId': 'FRA1', 'currentReplicationCount': 1, 'desiredReplicationCount': 1}]}, {'id': '17b49567-29e1-4d12-8337-9392ed6cb323', 'ipfs\_pin\_hash': 'QmZSMw1PARg1dDxaN2CoNbskraRXJo7trUsoUickrRgrTn', 'size': 148391, 'user\_id': '7be18edd-36fd-41c9-80ad-7a950eb7e104', 'date\_pinned': '2022-04-13T20:18:36.974Z', 'date\_unpinned': None, 'metadata': {'name': '6\_helloWeb3DeployMining.png', 'keyvalues': None}, 'regions': [{'regionId': 'FRA1', 'currentReplicationCount': 1, 'desiredReplicationCount': 1}]}]}

>>> myPic=pinata.pin\_list()['rows'][1]['ipfs\_pin\_hash']  
>>> myMeta=pinata.pin\_list()['rows'][0]['ipfs\_pin\_hash']  
>>> print(myPic)  
QmZSMw1PARg1dDxaN2CoNbskraRXJo7trUsoUickrRgrTn  
>>> print(myMeta)  
QmbdwcmzYacLZHLBAhAsaJm2aPV1bzBjFjweD1UgZzkEtA

해시를 이용해서 URI를 생성해 보자. 마지막 적힌 URL 이것이 TokenURI로 Solidity 프로그램에 적히게 된다.

>>> import requests  
>>> # Get our pinned item  
>>> #gateway="https://gateway.pinata.cloud/ipfs/"  
>>> gateway="https://ipfs.io/ipfs/"  
>>> IpfsHash=myMeta;  
>>> myUrl=gateway+IpfsHash;  
>>> #print(requests.get(url=myUrl).text)  
>>> print(myUrl)  
https://ipfs.io/ipfs/QmbdwcmzYacLZHLBAhAsaJm2aPV1bzBjFjweD1UgZzkEtA

# 6. NFT 개발

NFT는 어렵지 않게 만들 수 있다. NFT 표준이 정해져 있기 때문에, 심지어는 OpenZeppelin에서 제공하는 자동 생성기를 사용하면 그대로 사용할 수 있다 (https://docs.openzeppelin.com/contracts/4.x/wizard)

## 6.1 몇 줄로 완성되는 NFT

아주 간단한 NFT를 제작해보자. 제작할 때는 반드시 (1) NFT 명칭, (2) NFT 심볼, (3) tokenURI가 필요하다. 명칭과 심볼은 명확하지만 tokenURI는 메타데이터가 저장된 url을 말한다.

[파일명: src/MyNFT.sol]  
// SPDX-License-Identifier: MIT  
pragma solidity ^0.8.0;  
  
import "@openzeppelin/contracts/token/ERC721/ERC721.sol";  
//import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.4.0/contracts/token/ERC721/ERC721.sol";  
import "@openzeppelin/contracts/utils/Counters.sol";  
//import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.4.0/contracts/utils/Counters.sol";  
  
contract MyNFT is ERC721 {  
 using Counters for Counters.Counter;  
 Counters.Counter private \_tokenIds; //NFT id  
   
 constructor() ERC721("Hello DApp Screens", "JSL") {}  
 // override  
 // does not save toeknURI in the state var  
 function \_baseURI() internal pure override returns (string memory) {  
 return "https://ipfs.io/ipfs/QmbdwcmzYacLZHLBAhAsaJm2aPV1bzBjFjweD1UgZzkEtA";  
 }  
  
 function mint(address to) public returns(uint256) {  
 //require(\_tokenIds.current() < 3); //only 3 to mint  
 \_tokenIds.increment(); // add 1 by minting  
 uint256 newTokenId = \_tokenIds.current();  
 \_mint(to, newTokenId); // or \_safeMint(to, newTokenId);  
  
 return newTokenId;  
 }  
}

REMIX에서 코드를 작성하고, 테스트해서 함수들이 올바르게 작동하는지 확인하자. 그러고 나서 컴파일하고 abi, 바이트 코드를 생성하자. 위 코드 상단 import문의 http 경로는 바로 인식하지 못한다. 로컬의 경로로 @openzeppelin/를 재지정해서 아래와 같이 컴파일하자.

pjt\_dir> solc-windows.exe @openzeppelin/="C:\\Users\\jsl\\Code\\201711111\\node\_modules\\@openzeppelin\\" --optimize --combined-json abi,bin src/MyNFT.sol > src/MyNFT.json

### OpenZeppelin ERC 721 가져오기

OpenZeppelin은 ERC-721 표준을 준수하는 라이브러리를 제공하고 있다. 이미 설치한 @OpenZeppelin을 import문에 적용한다.

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";

### 상속

ERC721을 상속을 한다. ERC721은 OpenZeppelin에서 제공되는 컨트랙이다. 이 컨트랙을 상속받으면, NFT를 제작하는데 필요한 작업을 상속받아서 활용할 수 있어서 직접 코딩해야 하는 분량이 줄어들 뿐만 아니라, 앞서 설명한 바와 같이 NFT 표준을 준수하게 된다.

contract contract MyNFT is ERC721 {}

### Counters 라이브러리

위 코드에서 Counters와 관련한 코드를 부분적으로 가져와 붙여넣으면 아래와 같고, 이를 설명하자.

줄1 import "@openzeppelin/contracts/utils/Counters.sol";  
줄2 using Counters for Counters.Counter //Counters.Counter에 대해 라이브러리 함수를 사용  
줄3 Counters.Counter private \_tokenIds; //\_tokenIds를 Counters.Counter로 선언  
줄4 \_tokenIds.increment()

* 줄1에서 import문으로 @openzeppelin/contracts/utils/Counters.sol를 가져오고 있다. 그리고
* 줄2의 구문으로 Counters.Counter 데이터타입에 대해 데이터타입.함수() 형태로 사용하겠다는 뜻이다.
* 줄3에서는 Counters.Counter의 변수로서 \_tokenIds를 선언하고 있다.
* 줄4 이 라이브러리를 적용하여, Counters에 설정된 Counter변수로 정의된 변수에 대해 \_tokenIds.increment() 식으로 코딩하고 있다.

Counters 라이브러리를 보면 적용가능한 함수들이 정의되어 있는데, 아래는 그 일부를 가져와 보여주고 있다.

library Counters {  
 struct Counter { uint256 \_value; }  
 ...  
 function current(Counter storage counter) returns (uint256) { return counter.\_value; }  
 function increment(Counter storage counter) { counter.\_value += 1; }  
 }  
}

라이브러리를 적용하다보니 코드가 좀 복잡하다고 느낄 수 있지만, 의미는 명료하게 tokenIds를 하나 증가하는 것이다.

### mint 함수

mint() 함수에서는 내부용 함수인 \_mint()를 호출하는데, 다음과 같이 줄1에서 NFT 소유 개수를 하나 증가시키고, 줄2에서 소유권자를 할당한다.

줄1 \_balances[to] += 1; 개수 증가  
줄2 \_owners[tokenId] = to; \_owner에 새로운 소유주를 할당한다.

\_mint() 함수는 emit Transfer(address(0), to, tokenId)를 포함하고 있어서, 이벤트가 발생한다.

\_safeMint() 함수는 \_mint()와 동일하지만 안전을 위해 ERC721Receiver를 구현해 놓아야 한다.

\_checkOnERC721Received(address(0), to, tokenId, data),  
 "ERC721: transfer to non ERC721Receiver implementer"

### TokenURI의 저장

앞서 TokenURI의 메타데이터는 JSON형식으로 정의하였다. 여기에 이미지의 URL을 저장해 놓고 있어, 클릭하면 컨텐츠가 출력된다는 것을 기억하자.

TokenURI를 저장하기 위해, MyNFT의 \_baseURI() 함수를 재정의하고 있다. 이 함수는 아래와 같이 tokenURI(tokenId)에서 호출되고 URI를 출력할 수 있다.

function tokenURI(uint256 tokenId) public view virtual override returns (string memory) {  
 require(\_exists(tokenId), "ERC721Metadata: URI query for nonexistent token");  
 string memory baseURI = \_baseURI();  
 return bytes(baseURI).length > 0 ? string(abi.encodePacked(baseURI, tokenId.toString())) : "";  
}

그러니까 NFT는 URL을 가지고 있을 뿐이다. 사진이나 그림 등 디지털 자산 자체가 아니라, 그 메타데이터가 저장된 URL로 기록된 것이다.

## 6.2 URI를 저장하는 코드

ERC721URIStorage는 ERI721를 확장하여 URI를 저장하고 있다.

abstract contract ERC721URIStorage is ERC721 {  
 using Strings for uint256;  
 mapping(uint256 => string) private \_tokenURIs;  
 function tokenURI(uint256 tokenId) public view virtual override returns (string memory)  
 function \_setTokenURI(uint256 tokenId, string memory \_tokenURI) internal virtual  
 function \_burn(uint256 tokenId) internal virtual override  
}

ERC721은 tokenURI를 저장하지 않지만, 여기에서는 이를 실제 저장하고 있다. **ERC721.sol에서는 baseURI를 상태 변수에 저장하지 않고 메모리에서 읽도록** 하고 있는 것과 대비된다.

ERC721URIStorage는 tokenURI를 mapping(uint256 => string) private \_tokenURIs에 저장한다.

또한 IERC721Enumerable와 같은 인터페이스가 제공되고 있다. 잔고의 총개수를 읽으려면 이 인터페이스의 totalsupply()를 구현한다. 간단히 counter.current()를 실행하면 총개수를 알 수 있다.

### contract ... is ERC721, ERC721URIStorage 오류

contract ... is ERC721, ERC721URIStorage 이렇게 다중상속을 하면 'override' 오류가 발생한다. 따라서 아래 예와 같이, ERC721URIStorage만 상속을 받도록 한다. 그러면 ERC721URIStorage가 ERC721을 이미 상속받았기 때문에 오류없이 재정의할 수 있다.

[파일명: src/MyNFTuri.sol]  
// SPDX-License-Identifier: MIT  
pragma solidity ^0.8.0;  
  
import "@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol";  
//import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.4.0/contracts/token/ERC721/extensions/ERC721URIStorage.sol";  
import "@openzeppelin/contracts/utils/Counters.sol";  
//import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.4.0/contracts/utils/Counters.sol";  
  
contract MyNFT is ERC721URIStorage {  
 using Counters for Counters.Counter;  
 Counters.Counter private \_tokenIds; //NFT id  
  
 constructor() ERC721("MyNFT", "JSL") {}  
  
 function mint(address to, string memory tokenURI) public returns(uint256) {  
 \_tokenIds.increment(); // add 1 by minting  
 uint256 newTokenId = \_tokenIds.current();  
 \_mint(to, newTokenId); // or \_safeMint(to, newTokenId);  
 \_setTokenURI(newTokenId, tokenURI); // need to import ERC721URIStorage  
  
 return newTokenId;  
 }  
}

pjt\_dir> solc-windows.exe @openzeppelin/="C:\\Users\\jsl\\Code\\201711111\\node\_modules\\@openzeppelin\\" --optimize --combined-json abi,bin src/MyNFTuri.sol > src/MyNFTuri.json

## 6.3 Ownable 추가한 코드

### 토큰의 소유권

ERC721에서는 \_owners를 설정하였고, tokenId 소유자를 의미한다. mint하면서 \_owners가 다음과 같이 정의된다.

mapping(uint256 => address) private \_owners;

반면에 Ownable에서 설정하는 Owner는 msg.sender, 즉 컨트랙을 생성하는 최초의 컨트랙 소유주를 말한다.

address public owner;

이 Owner의 소유권을 설정하고, 이전하는 기능을 제공한다.

### onlyOwner

상속을 통해서 구현을 하면 modifier onlyOwner라는 수식어를 사용할 수 있게 된다. 피상속을 받으면 함수를 구현하면서 onlyOwner라는 수식어를 붙여 소유권을 제한할 수 있게 된다.

getOwner() 함수는 Owner를 반환한다. 주의할 점은 Owner를 설정하지 않았는데도 불구하고 그 값을 알 수 있다. Ownable에서 그 값을 가지고 있기 때문이다.

### transferToPay

ERC721에 정의된 \_exists() 함수는 tokenId에 해당하는 소유자가 있는지 확인하는 함수이다.

function \_exists(uint256 tokenId) internal view virtual returns (bool) {  
 return \_owners[tokenId] != address(0);  
}

[파일명: src/MyNFTUriOwner.sol]  
// SPDX-License-Identifier: MIT  
pragma solidity ^0.8.0;  
  
import "@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol";  
//import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.4.0/contracts/token/ERC721/extensions/ERC721URIStorage.sol";  
import "@openzeppelin/contracts/utils/Counters.sol";  
//import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.4.0/contracts/utils/Counters.sol";  
import "@openzeppelin/contracts/access/Ownable.sol";  
//import "https://github.com/nibbstack/erc721/blob/2.6.1/src/contracts/ownership/ownable.sol";  
  
//Ownable added: minToPay, transferToPay  
contract MyNFTUriOwner is ERC721URIStorage, Ownable {  
 using Counters for Counters.Counter;  
 Counters.Counter private \_tokenIds; //NFT id  
  
 mapping(uint256 => uint256) public tranPrice;  
  
 event Transfer(address from, address to, uint256 \_tokenId,  
 string tokenURI, uint256 tranPrice);  
  
 constructor() ERC721("MyNFT", "JSL") {}  
  
 function mintWithURI(address to, string memory tokenURI) public returns(uint256) {  
 \_tokenIds.increment(); // add 1 by minting  
 uint256 newTokenId = \_tokenIds.current();  
 \_mint(to, newTokenId); // or \_safeMint(to, newTokenId);  
 \_setTokenURI(newTokenId, tokenURI); // need to import ERC721URIStorage  
 return newTokenId;  
 }  
 function mintWithIdURI(uint256 \_id, address to, string memory tokenURI) public returns(uint256) {  
 \_mint(to, \_id);  
 \_setTokenURI(\_id, tokenURI);  
 return \_id;  
 }  
 //Note who calls this function. Does not make sense if owner is the one who pays  
 function mintWithURIPrice(address to, string memory tokenURI, uint256 \_p)  
 public payable onlyOwner returns(uint256) {  
 require(msg.value == \_p, "Not Enough Ether");  
 uint256 newTokenId = mintWithURI(to, tokenURI);  
 tranPrice[newTokenId] = \_p;  
 return newTokenId;  
 }  
 function getTotalSupply() public view returns(uint256) { return \_tokenIds.current(); }  
 function getOwner() view public returns(address) {  
 return owner(); //function of the Ownerable contract  
 }  
  
 // msg.value transferred to this contract!!  
 function transferToPay(address from, address to, uint256 \_tokenId) public payable {  
 require(\_exists(\_tokenId), "Error: TokenId not owned");  
 require(msg.value >= tranPrice[\_tokenId], "Error: Token costs more");  
 \_transfer(from, to, \_tokenId);  
 emit Transfer(from, to, \_tokenId, tokenURI(\_tokenId), tranPrice[\_tokenId]);  
 }  
 function burn(uint256 \_tokenId) public { \_burn(\_tokenId); }  
}

여기서는 방금 완성한 NFT를 컴파일해보자.

pjt\_dir> solc-windows.exe @openzeppelin/="C:\\Users\\jsl\\Code\\201711111\\node\_modules\\@openzeppelin\\" --optimize --combined-json abi,bin src/MyNFTUriOwner.sol > src/MyNFTUriOwner.json

## 6.4 단계 3: 배포

우선 사설망에 배포하고 나중에 공중망으로 해보자. 총 13개 abi, 바이트 코드가 파일에 포함되어 있으며, 세어보면 12번째 'src/MyNFTUriOwner.sol:MyNFTUriOwner'가 바로 우리가 배포할 대상이다.

컨트랙 명을 읽어오는 프로그램을 작성해보자.

[파일명: src/MyNFTUriOwnerDeploy.js]  
var Web3 = require('web3');  
var \_abiBinJson = require('./MyNFTUriOwner.json'); //importing a javascript file  
  
var web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8345"));  
  
contractName=Object.keys(\_abiBinJson.contracts);  
//console.log(contractName);  
console.log(contractName[12]);

실행하면, contractName[12]의 컨트랙명을 출력하고 있다.

pjt\_dir> node src/MyNFTUriOwnerDeploy.js  
  
src/MyNFTUriOwner.sol:MyNFTUriOwner

생정자에서 명칭과 심볼은 이미 적어 놓았기 때문에, 별도로 설정하지 않고 있다. REMIX에서 해보면, gas가 상당히 필요하다는 것을 알 수 있다. 1700000으로 해도 gas limit 오류가 발생하고 있다.

[파일명: src/MyNFTUriOwnerDeploy.js]  
var Web3 = require('web3');  
var \_abiBinJson = require('./MyNFTUriOwner.json'); //importing a javascript file  
  
var web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8345"));  
  
contractName=Object.keys(\_abiBinJson.contracts);  
console.log("- contract name: ", contractName[12]);  
\_abiArray=\_abiBinJson.contracts[contractName[12]].abi;  
//\_abiArray=JSON.parse(JSON.stringify(\_abi));  
//\_abiArray=JSON.parse(\_abi); //JSON parsing needed!! //SyntaxError: Unexpected token o in JSON at position 1  
\_bin="0x"+\_abiBinJson.contracts[contractName[12]].bin;  
  
async function deploy() {  
 const accounts = await web3.eth.getAccounts();  
 console.log("Deploying the contract from " + accounts[0]);  
 var deployed = await new web3.eth.Contract(\_abiArray)  
 .deploy({data: \_bin})  
 .send({from: accounts[0], gas: 2000000}, function(err, transactionHash) {  
 if(!err) console.log("hash: " + transactionHash);   
 })  
 console.log("---> The contract deployed to: " + deployed.options.address)  
}  
deploy()

배포한 결과, hash와 컨트랙 주소를 출력하고 있다.

pjt\_dir> node src/MyNFTUriOwnerDeploy.js  
  
- contract name: src/MyNFTUriOwner.sol:MyNFTUriOwner  
Deploying the contract from 0xE4F45Ccc21de92cCB52C0dF5f0325c71986FB132  
hash: 0x4c4c456f372fdd4eaf690a4031895274ab86875d9322c407c3323908e8b98b53  
---> The contract deployed to: 0x04F2A101720621B32cFF15cFfD9C15CbBE1cC780

## 6.5 단계 4: 사용

NFT를 발행하고, 소유권을 이체하여 보자.

[파일명: src/MyNFTUriOwnerUse.js]  
var Web3=require('web3');  
var \_abiBinJson = require('./MyNFTUriOwner.json'); //importing a javascript file  
  
var web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8345"));  
//var web3 = new Web3(new Web3.providers.WebsocketProvider("ws://localhost:8345"));  
contractName=Object.keys(\_abiBinJson.contracts);  
console.log("- contract name: ", contractName[12]);  
\_abiArray=\_abiBinJson.contracts[contractName[12]].abi;  
//\_abiArray=JSON.parse(JSON.stringify(\_abi));  
//\_abiArray=JSON.parse(\_abi); //JSON parsing needed!! //SyntaxError: Unexpected token o in JSON at position 1  
\_bin="0x"+\_abiBinJson.contracts[contractName[12]].bin;  
var \_nft = new web3.eth.Contract(\_abiArray,"0x04F2A101720621B32cFF15cFfD9C15CbBE1cC780");  
var event = \_nft.events.Transfer({fromBlock: 0}, function (error, result) {  
 if (!error) {  
 console.log("Event fired: " + JSON.stringify(result.returnValues));  
 }  
});  
  
async function doIt() {  
 const accounts = await web3.eth.getAccounts();  
 console.log("(0) Account: " + accounts[0]);  
 var owner = await \_nft.methods.getOwner().call();  
 console.log("(1) owner: "+owner);  
 console.log("(2) name: "+await \_nft.methods.name().call());  
 console.log("(3) symbol: "+await \_nft.methods.symbol().call());  
  
 var uri = "https://ipfs.io/ipfs/QmbdwcmzYacLZHLBAhAsaJm2aPV1bzBjFjweD1UgZzkEtA";  
 var uri2= "http://example.com";  
 var addrTo1 = accounts[1];  
 var addrTo2 = accounts[2];  
 console.log("(4) minting from "+owner+" to "+addrTo1);  
 await \_nft.methods.mintWithURI(addrTo1,uri).send({from: accounts[0], gas: 1000000});  
 console.log("(5) minting from "+owner+" to "+addrTo2);  
 await \_nft.methods.mintWithURI(addrTo2,uri).send({from: accounts[0], gas: 1000000});  
 var \_tid = 1;  
 //console.log("(6) [reverted] minting with a duplicate tokenId 1"+" to "+addrTo1);  
 //await \_nft.methods.mintWithIdURI(\_tid,addrTo1,uri).send({from: accounts[0], gas: 1000000});  
 console.log("(7) Total number minted: "+await \_nft.methods.getTotalSupply().call());  
 console.log("(8) balanceOf "+owner+": "+await \_nft.methods.balanceOf(owner).call());   
 console.log("(8) balanceOf "+addrTo1+": "+await \_nft.methods.balanceOf(addrTo1).call());  
 console.log("(8) balanceOf "+addrTo2+": "+await \_nft.methods.balanceOf(addrTo2).call());  
 console.log("(9) ownerOf tokenId "+\_tid+": "+await \_nft.methods.ownerOf(\_tid).call());  
 console.log("(9) tokenURI of tokenId "+\_tid+": "+await \_nft.methods.tokenURI(\_tid).call());  
 console.log("(10) [TRANSFER ELSE OWNERSHIP IN NEED OF APPROVAL] owner transfers from addrTo1 to addrTo2");  
 console.log("(10) Who gets approved: "+await \_nft.methods.getApproved(\_tid).call());  
 console.log("(10) addrTo1 approves owner to transfer...");  
 await \_nft.methods.approve(owner, \_tid).send({from: addrTo1, gas: 1000000});  
 console.log("(10) Now gets approved (addrTo1 --> owner): "+await \_nft.methods.getApproved(\_tid).call());  
 console.log("(10) then enables owner to transfer...")  
 await \_nft.methods.transferFrom(addrTo1, addrTo2, \_tid).send({from: owner, gas: 1000000});  
 console.log("(10) ownerOf after transferred: "+await \_nft.methods.ownerOf(\_tid).call());  
 console.log("(11) [TRANSFER SELF OWNERSHIP] transfers mine to else --> no need of approval");  
 await \_nft.methods.transferFrom(addrTo2, accounts[3], \_tid).send({from: addrTo2, gas: 1000000});  
 console.log("(11) balanceOf "+owner+": "+await \_nft.methods.balanceOf(owner).call());   
 console.log("(11) balanceOf "+addrTo1+": "+await \_nft.methods.balanceOf(addrTo1).call());  
 console.log("(11) balanceOf "+addrTo2+": "+await \_nft.methods.balanceOf(addrTo2).call());  
 console.log("(11) balanceOf "+accounts[3]+": "+await \_nft.methods.balanceOf(accounts[3]).call());  
 console.log("(11) ownerOf after transferred: "+await \_nft.methods.ownerOf(\_tid).call());  
 console.log("(12) Now burns: "+await \_nft.methods.burn(\_tid).send({from: accounts[3], gas: 1000000}));  
 console.log("(12) balanceOf "+accounts[3]+": "+await \_nft.methods.balanceOf(accounts[3]).call());  
 //process.exit(1); //force exit  
}  
doIt()

가능한 함수를 모두 호출하기 때문에 코드가 길어지고 있다. 단말에서 다음을 실행하자.

pjt\_dir> node src/MyNFTUriOwnerUse.js  
  
- contract name: src/MyNFTUriOwner.sol:MyNFTUriOwner  
(0) Account: 0xE4F45Ccc21de92cCB52C0dF5f0325c71986FB132  
(1) owner: 0xE4F45Ccc21de92cCB52C0dF5f0325c71986FB132  
(2) name: MyNFT  
(3) symbol: JSL  
(4) minting from 0xE4F45Ccc21de92cCB52C0dF5f0325c71986FB132 to 0x421A4FBD73Ba4E829aF760A375f5A912538393D6  
(5) minting from 0xE4F45Ccc21de92cCB52C0dF5f0325c71986FB132 to 0x1BFBEf55C90001746deC61589b216A6d2e6e30e4  
(7) Total number minted: 2  
(8) balanceOf 0xE4F45Ccc21de92cCB52C0dF5f0325c71986FB132: 0  
(8) balanceOf 0x421A4FBD73Ba4E829aF760A375f5A912538393D6: 1  
(8) balanceOf 0x1BFBEf55C90001746deC61589b216A6d2e6e30e4: 1  
(9) ownerOf tokenId 1: 0x421A4FBD73Ba4E829aF760A375f5A912538393D6  
(9) tokenURI of tokenId 1: https://ipfs.io/ipfs/QmbdwcmzYacLZHLBAhAsaJm2aPV1bzBjFjweD1UgZzkEtA  
(10) [TRANSFER ELSE OWNERSHIP IN NEED OF APPROVAL] owner transfers from addrTo1 to addrTo2  
(10) Who gets approved: 0x0000000000000000000000000000000000000000  
(10) addrTo1 approves owner to transfer...  
(10) Now gets approved (addrTo1 --> owner): 0xE4F45Ccc21de92cCB52C0dF5f0325c71986FB132  
(10) then enables owner to transfer...  
(10) ownerOf after transferred: 0x1BFBEf55C90001746deC61589b216A6d2e6e30e4  
(11) [TRANSFER SELF OWNERSHIP] transfers mine to else --> no need of approval  
(11) balanceOf 0xE4F45Ccc21de92cCB52C0dF5f0325c71986FB132: 0  
(11) balanceOf 0x421A4FBD73Ba4E829aF760A375f5A912538393D6: 0  
(11) balanceOf 0x1BFBEf55C90001746deC61589b216A6d2e6e30e4: 1  
(11) balanceOf 0xC6b4880724ab48D8e3Cc52Cc72C57aaaBE7a4301: 1  
(11) ownerOf after transferred: 0xC6b4880724ab48D8e3Cc52Cc72C57aaaBE7a4301  
(12) Now burns: [object Object]  
(12) balanceOf 0xC6b4880724ab48D8e3Cc52Cc72C57aaaBE7a4301: 0

출력된 결과를 설명해보자.

* (1~3)에서는 이름, symbol, owner를 출력하고 있다. .getOwner()는 소유자를 확인하는 함수로 owner()와 동일한 기능이다.
* (4~5)에서는 mintWithURI() 함수에 이전될 계정주소와 tokenURI "https://ipfs.io/ipfs/QmbdwcmzYacLZHLBAhAsaJm2aPV1bzBjFjweD1UgZzkEtA" 를 넘겨주고 있다.
  1. 생성된 NFT의 총 갯수 2가 출력되고 있다.
  2. balanceOf 주소를 넣으면 가지고 있는 NFT 갯수, 처음에는 당연히 0이 출력된다. 0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db는 2출력
  3. ownerOf(tokenId)의 owner주소 앞서 balanceOf에 입력한 수를 넣으면 주소가 출력
  4. transfer함수는 token 소유자가 다른 주소에게 소유권을 이전하는 함수이다. 다른 사람이 하려면 approve로 그 허가권을 획득해야 가능하다.
  5. 자신의 소유권은 그대로 이전할 수 있지만, 타인의 소유권을 이전하려면 허가를 받고 나서야 가능하다.
  6. burn하고 나면 \_owners[tokenId]가 삭제된다. 따라서 소유주를 출력할 수 없게 된다.

코드에 설정한 바와 같이, 소유권이 변경될 경우 Transfer(from, to, tokenId) 이벤트가 발생한다.

하나의 자바스크립트로 병합할 수 있지만, 웹소켓이 이벤트를 대기하기 때문에 출력을 확인하기 불편할 수 있다. 별도의 명령창에 이벤트가 포함된 다음 프로그램을 실행해 놓으면 된다.

[파일명: src/MyNFTUriOwnerUseEventOnly.js]  
var Web3=require('web3');  
var \_abiBinJson = require('./MyNFTUriOwner.json');  
  
//var web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8345"));  
var web3 = new Web3(new Web3.providers.WebsocketProvider("ws://localhost:8345"));  
contractName=Object.keys(\_abiBinJson.contracts);  
console.log("- contract name: ", contractName[12]);  
\_abiArray=\_abiBinJson.contracts[contractName[12]].abi;  
//\_abiArray=JSON.parse(JSON.stringify(\_abi));  
//\_abiArray=JSON.parse(\_abi); //JSON parsing needed!! //SyntaxError: Unexpected token o in JSON at position 1  
\_bin="0x"+\_abiBinJson.contracts[contractName[12]].bin;  
var \_nft = new web3.eth.Contract(\_abiArray,"0x04F2A101720621B32cFF15cFfD9C15CbBE1cC780");  
var event = \_nft.events.Transfer({fromBlock: 0}, function (error, result) {  
 if (!error) {  
 console.log("Event fired: " + JSON.stringify(result.returnValues));  
 }  
});

Transfer 이벤트가 발생할 때마다 다음과 같은 출력을 볼 수 있다.

pjt\_dir> node src\MyNFTUriOwnerUseEventOnly.js  
- contract name: src/MyNFTUriOwner.sol:MyNFTUriOwner  
Event fired: {"0":"0x0000000000000000000000000000000000000000","1":"0x421A4FBD73Ba4E829aF760A375f5A912538393D6","2":"1","from":"0x0000000000000000000000000000000000000000","to":"0x421A4FBD73Ba4E829aF760A375f5A912538393D6","tokenId":"1"}  
Event fired: {"0":"0x0000000000000000000000000000000000000000","1":"0x1BFBEf55C90001746deC61589b216A6d2e6e30e4","2":"2","from":"0x0000000000000000000000000000000000000000","to":"0x1BFBEf55C90001746deC61589b216A6d2e6e30e4","tokenId":"2"}  
Event fired: {"0":"0x421A4FBD73Ba4E829aF760A375f5A912538393D6","1":"0x1BFBEf55C90001746deC61589b216A6d2e6e30e4","2":"1","from":"0x421A4FBD73Ba4E829aF760A375f5A912538393D6","to":"0x1BFBEf55C90001746deC61589b216A6d2e6e30e4","tokenId":"1"}  
Event fired: {"0":"0x1BFBEf55C90001746deC61589b216A6d2e6e30e4","1":"0xC6b4880724ab48D8e3Cc52Cc72C57aaaBE7a4301","2":"1","from":"0x1BFBEf55C90001746deC61589b216A6d2e6e30e4","to":"0xC6b4880724ab48D8e3Cc52Cc72C57aaaBE7a4301","tokenId":"1"}  
Event fired: {"0":"0xC6b4880724ab48D8e3Cc52Cc72C57aaaBE7a4301","1":"0x0000000000000000000000000000000000000000","2":"1","from":"0xC6b4880724ab48D8e3Cc52Cc72C57aaaBE7a4301","to":"0x0000000000000000000000000000000000000000","tokenId":"1"}

# 7. NFT를 공중망에 배포해보자

## 직접 공중망에 연결

공중망에 배포하려면, 자신이 geth 단말에서 직접 연결해야 한다. 앞서 해보았던 것과 같이, geth --testnet으로 테스트망에 연결하여 배포하는 것이다.

## 암호지갑을 통해서 공중망에 연결

암호지갑 메타마스크(MetaMask)에 연결해서 배포할 수 있다. 그렇게 하면 REMIX에서 메타마스크에 연결한 후 (Injected Web3), 내가 가진 키를 읽어낼 수 있게 된다. 이 때 REMIX를 https가 아니라 http로 사용해야 한다.

절차를 하나씩 나열해 보면:

* 메타마스크에서 Ropsten을 선택한다.
* 메타마스크에 Ropsten 계정이 없으면 생성을 선택하고, 키를 복사해 놓는다.
* 충전을 한다. 충전서비스를 연결해서 faucet.metamask.io에서 복사해 놓은 키를 붙여넣기 하고 충전을 대기한다. faucet.ropsten.be는 10초에 0.3 Ether를 흘려보내고 있다. 보통 많은 사용자가 대기하고 있어 시간이 걸린다.
* REMIX에 가서 Injected Web3를 선택하면 메타마스크와 연결하는 팝업 창이 뜬다. 연결하고 나면 좌상단에 "연결됨"이 뜬다.

## 제 3자 서비스 Infura 에 연결

지금까지 우리가 해왔던 방식으로 스크립트를 작성해서 배포하려면, 제3자 서비스 Infura, QuickNode, Alchemy 등을 이용하면 된다. URL만 적어주면 별도의 작업없이 쉽게 연결된다.

웹브라우저에서는 브라우저 확장앱인 MetaMask에서 직접 읽어올 수 있지만, 노드에서는 그렇게 하지 못한다. 따라서 제3자 서비스제공자에서 키를 읽어올 수 없으므로 키를 하드코딩해야 한다.

Infura는 블록체인 노드 provider이고,블록체인과 IPFS에 대한 API를 제공한다. Web3 서비스를 제공한다. Infura를 사용하지 않으면, 자신의 컴퓨터에 블록체인을 항상 동기화 해야 하는데, 시간이 오래 걸릴 뿐만 아니라, 컴퓨터에 부하가 걸릴 수 있다.

Infura (https://infura.io) 에 접속한 후, 먼저 회원가입을 한다. 처음에는 아무 프로젝트도 없는 상태이고, 새 프로젝트를 만들어주어야 한다. 예를 들어, TEST라는 이름의 프로젝트를 만들면, 바로 API KEY를 생성해준다.

infura에 로그인해서 자신의 PROJECT SETTINGS를 열어보면, PROJECT ID와 PROJECT SECRET를 발견할 수 있다. 그 우측에 보면 curl 명령어로 자신의 네트워크에 대해 블록번호를 아래와 같이 테스트해볼 수 있다.

pjt\_dir> curl https://mainnet.infura.io/v3/65b8df8bc3e84ae29d55984be7dc1d11 \  
-X POST \  
-H "Content-Type: application/json" \  
-d '{"jsonrpc":"2.0","method":"eth\_blockNumber","params":[],"id":1}'  
  
{"jsonrpc":"2.0","id":1,"result":"0xe2bbe1"} //블록번호를 출력한다.

curl 명령어에 더불어 적고 있는 스위치를 보자.

* -X: 함수 종류를 적는다. 예제에서는 POST 함수라는 의미.
* -H: 헤더를 적는다. 예제에서는 Content-Type이 JSON 형식이라는 의미.
* -d: 전송될 데이터를 적는다. 예제에서는 JSON형식으로 적는다.

앞서 만든 infura에 접속하여 테스트하는 프로그램을 작성해보자.

[파일명: scripts/testRopsten.js]  
var Web3=require('web3');  
  
//var web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8345"));  
var web3 = new Web3(new Web3.providers.HttpProvider('https://ropsten.infura.io/v3/65b8df8bc3e84ae29d55984be7dc1d11')); // ropsten  
// var web3 = new Web3(new Web3.providers.HttpProvider('https://mainnet.infura.io')); // main  
  
async function doIt() {  
 console.log("Account: " + await web3.eth.getAccounts()); //empty  
 //console.log("request Account: " + await web3.eth.requestAccounts()); --> Error reading accounts  
 console.log("Balance E1931: " + await web3.eth.getBalance("0xE1931BF7ec2EcBD176C9b1852b64753878A316c5"));  
 console.log("Balance 591c8: " + await web3.eth.getBalance("0x591c86219c77B0Be11a0D57D42DaB6915E2E6f92"));  
 console.log("block number: "+await web3.eth.getBlockNumber());  
}  
  
doIt()

Infura는 API만을 제공하기 때문에, 계정은 없고 따라서 아무 것도 출력하지 않고 있다. infura에 연결한 후, 계정의 잔고를 출력하고 있다.

pjt\_dir> node scripts/testRopsten.js  
  
Account:   
Balance E1931: 1000000000000000000  
Balance 591c8: 3000000000000000000  
block number: 12376559

# 8. 나의 NFT를 판매해 보자

NFT를 제작하고 나서 이대로 끝내기에는 뭔가 미진하다. 이제 자신이 만든 NFT를 판매해보자.

## 8.1 마켓플레이스에 등록

다음과 같은 사이트에서는 온라인에서 NFT를 사고 팔 수 있는 장터가 제공되고 있다. 버튼 클릭 몇 번만 하면 간단하게 NFT를 제작할 수 있기도 하다.

* Opensea: 디지털 아이템이나 암호화된 수집품 등을 사고 팔 수 있는 마켓플레이스. 2017년 12월 개시되어 현재 전 세계 NFT의 대부분이 거래되고 있다.
* Rarible: 블록체인을 통해 디지털 소장품을 만들어 사고 팔 수 있는 마켓플레이스
* Async: 예술 작품을 토큰화하여 사고 팔 수 있는 마켓플레이스
* Mintable: 블록체인 아이템을 사고 팔 수 있는 마켓플레이스
* SuperRare: 개별 디지털 예술품을 소장하고 거래할 수 있는 마켓플레이스

국내에서는 가상 자산 거래소인 업비트, 빗썸 등 모두 NFT 시장 기능을 제공하고 있다. 뿐만 아니라 카카오, 네이버, 게임사 위메이드, 카카오게임즈, 크래프톤, 컴투스홀딩스 등 게임사들도 'P2E(Play to Earn, 돈을 벌 수 있는 게임)'시장을 겨냥해 NFT 거래소 구축에 나섰다. 다만 해외 거래소에서는 누구나 자신의 디지털자산을 NFT로 발행할 수 있지만 국내 거래소에서는 창작자로서 자격을 부여받는 등의 인증이 필요하다.

## 8.2 공중망에 등록

Rinkeby 테스트망에 배포하면 etherscan에서 볼 수 있다. 오픈씨에서도 확인할 수 있다 (https://testnets.opensea.io/)

ERC721을 이더리움 메인넷에 배포하면, 바로 오픈씨에서 매매가능하다 (https://opensea.io/get-listed).

ERC721을 작성해서 공중망에 배포하면, 바로 NFT시장에서는 신규토큰의 생성 이벤트의 발생을 금세 알아챈다. 이 이벤트는 로그를 모니터링하면 알게 된다. 그 로그는 누구나 볼 수 있기 때문에 오픈씨에서는 NFT의 소유자 주소, 토큰번호 등을 알게 된다. ERC721이 공중망에 배포되면 오픈씨나 Rarible이 알게 되는 이치이다.

## 8.3 코드 없이 등록

온라인 제작 사이트를 활용하면, 코딩을 하지 않고도 NFT를 제작할 수 있다. 우선 오픈씨 사이트(https://opensea.io/)에 로그인하고 접속한다.

NFT를 생성하기 위해서는 반드시 지갑을 연동해야 한다. 여기서는 메타마스크로 연결하여, mainnet에 개설된 자신의 주소를 선택한다.

자신의 지갑에 연결되면, 그 다음부터는 간단하다. 상품을 등록해보자. 파일 형식은 JPG, PNG, GIF, MP4, WAV 등이며, 최대 업로드 가능한 크기는 100MB다. 상품을 등록할 때 거래건별 2.5%의 수수료가 발생한다. 판매자는 등록된 상품에 대해 서비스 수수료 2.5%를 제외한 금액을 받는다.

## 연습문제

1. NFT는 어떤 저작물의 소유를 증명하지만, 블록체인 상에 공개적으로 존재하기 때문에 소유권을 누구나 이전할 있다. OX로 답하시오.
2. NFT는 디지털 자산이 될 수 있는 미술품, 부동산, 복권, 주식 어떤 것이든 대상이 될 수 있고, 말 그대로 Non Fungible, NFT 상호간에 교환이 될 수 없다는 것이다. OX로 답하시오.
3. 최초의 NFT는 2014년 5월 Kevin McCoy가 만든 "Quantum"이다. OX로 답하시오.
4. NFT를 제작하기 위해 준수해야 할 이더리움의 표준을 적으시오.
5. 계정주소의 보유 NFT 갯수를 나타내는 변수를 선언하는 코드를 작성하시오.
6. toeknId=1의 NFT가 저장된 URI를 반환하는 함수를 호출하시오.
7. ERC721을 이더리움 메인넷이나 테스트망에 배포하면, 매매가능하다. OX로 답하시오.
8. IFPS는 클라우드 저장소로서, 파일을 저장하고 블록체인과는 무관하다. OX로 답하시오.
9. NFT를 제작하기 위해 \_mint() 함수를 호출한다. 이 함수에서 하는 작업을 적으시오.
10. 오늘 일자가 보이는 화면을 2개 캡쳐한 이미지A, 이미지B를 NFT로 제작하시오. 컨트랙 명칭은 MyNFT로 하고, NFT와 관련한 정보를 저장하기 위해 아래 변수를 참조하시오.

* mapping(uint256 => Item) public tokenIdToItem 으로 선언하고 저장.
* tokenId에 대해 (1) 원소유자주소(즉 owner), (2) 현소유자주소, (3) 가격 (없으면 0), (4) tokenURI, (5) 현시각을 저장.

그리고 다음 함수를 구현하시오.

* function mintWithURI(address to, string memory tokenURI) public returns(uint256)
  + 소유자주소, tokenURI를 입력하고 mint하는 함수
  + 내부함수 \_mint()를 호출해서 처리
* function mintWithIdURI(uint256 \_id, address to, string memory tokenURI) public returns(uint256)
  + tokenId, 소유자주소, tokenURI를 입력하고 mint하는 함수
  + id가 이미 존재하는지 코딩하지 않아도 됨. 중복이면 자동으로 실패하는 로직을 내부적으로 이미 가지고 있다.
* function myTransfer(address from, address to, uint256 \_tokenId) public
  + from에서 to로 \_tokenId를 이전하는 함수
  + transferFrom 또는 내부함수 \_transfer를 호출해서 처리
* function setTokenIdToItem(uint256 \_id, address \_o, address \_to, uint256 \_p, string memory \_uri, uint256 \_t) public
  + tokenIdToItem 항목에 저장하는 함수
* getTokenIdToItem(uint256 tokenId) public view returns (address, address, uint256, string memory, uint256)
  + tokenIdToItem 항목을 출력
* getItemsLength() 함수: Item 개수 출력
* getOwner() 함수: 컨트랙의 소유주 출력
* getTokenURI(uint256 tokenId) 함수: tokenId의 tokenURI 출력
* getTotalSupply() 함수: 현재 tokenId가 총잔고이다.

NFT를 작성하고 아래 문항을 풀어보시오. (1)~(5)는 REMIX에서 실행하고, 주소0은 account[0], 주소1은 accounts[1], 주소2는 accounts[2]...으로 한다.

* 1. REMIX에서 mintWithURI(주소1, 이미지A의 URI) 함수의 성공 로그를 붙여 넣는다.
  2. REMIX에서 mintWithURI(주소2, 이미지B의 URI) 함수의 성공 로그를 붙여 넣는다.
  3. REMIX에서 mintWithIdURI(tokenId=1, 주소1, 이미지A의 URI) 함수의 실패 로그를 붙여 넣는다.
  4. REMIX에서 ownerOf(tokenId=1) 함수의 성공 로그를 붙여 넣는다.
  5. REMIX에서 transferFrom(tokenId=1, 주소1, 주소2) 함수의 성공 로그를 붙여 넣는다.
  6. 로컬에서 컴파일하세요.

다음 (7)부터는 노드 스크립트를 실행하여 풀어보시오 (ganache@8345 백그라운드 실행). 단, 테스트 실행하면서 소유이전을 반복하면 오류가 발생할 수 있으니 주의한다. 아예 배포주소를 새롭게 받아서 하면 처음부터 시작하게 되니까 당연히 오류가 사라질 수 있다.

* 1. ganache에서 배포하고, 컨트랙 배포주소 출력.
  2. 주소0에서 주소1에게 발행하세요. mintWithURI(주소1, 이미지A의 URI) 출력예시 --> (8) minting from 0xE4F45... to 0x421A4...
  3. 주소0에서 주소2에게 발행. mintWithURI(주소1, 이미지B의 URI) 출력예시 --> (9) minting from 0xE4F45... to 0x1BFBE...
  4. Item 개수를 출력. 현재 mint된 token개수, 2가 출력되어야 함. 출력예시 --> (10) Total number minted: 2
  5. 주소0, 주소1, 주소2의 잔고개수 출력. 출력예시 -->

1. balanceOf 0xE4F45...: 0
2. balanceOf 0x421A4...: 1
3. balanceOf 0x1BFBE...: 1
   1. tokenId 1의 소유권자 및 TokenURI 출력. 출력예시 -->
4. ownerOf tokenId 1: 0x421A4...
5. tokenURI of tokenId 1: https://ipfs.io/ipfs/QmbdwcmzYacLZHLBAhAsaJm2aPV1bzBjFjweD1UgZzkEtA
   1. getTokenIdToItem(tokenId=1)을 출력.
   2. owner가 tokenId 1를 주소1에서 주소2로 이전. tokenIdToItem도 당연히 수정되어야 한다. 그리고 변경된 tokenId 1의 소유주를 출력 (tokenIdToItem에서 읽어옴) 출력예시 --> (14) ownerOf tokenId=1 after transferred: 0x1BFBE...
   3. 주소2가 자신의 토큰(tokenId=2)을 주소3으로 소유권이전. 그리고 변경된 tokenId 2의 소유주를 출력. 출력예시 --> (15) ownerOf tokenId=2 after transferred: 0xC6b48...
   4. tokenId 2를 제거하세요 (burn). 제거되었는지 잔고를 출력. 출력예시 --> (16) balanceOf 0xC6b48...: 0
   5. 소유권이전 시 자동으로 Transfer(from, to, tokenId) 이벤트가 발생한다. tokenId 1번에 대해 발생한 이벤트를 출력하시오. 출력예시 ---> Event fired: {"0":"0xC6b4880724ab48D8e3Cc52Cc72C57aaaBE7a4301", "1":"0x0000000000000000000000000000000000000000", "2":"1", "from":"0xC6b4880724ab48D8e3Cc52Cc72C57aaaBE7a4301", "to":"0x0000000000000000000000000000000000000000", "tokenId":"1"}
   6. 현재 테스트넷의 블록넘버를 구하시오.
   7. 테스트넷에 배포하고 주소를 구하시오. REMIX 또는 스크립트를 작성해서 실행해도 된다.