한 번에 끝내는
블록체인 개발 A to Z

Chapter 2

Blockchain 2.0 - Ethereum

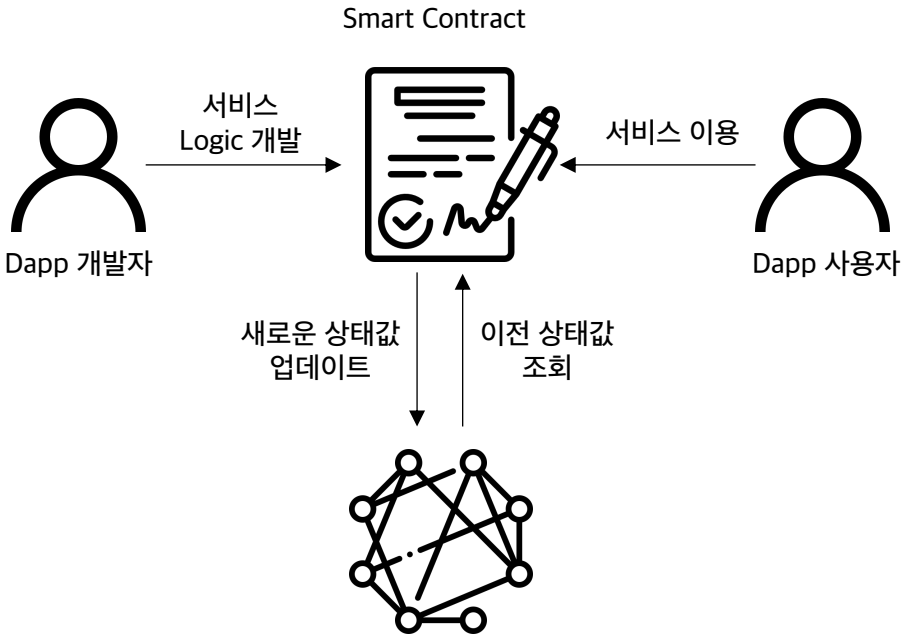
Chapter 2

Blockchain 2.0 - Ethereum

Smart Contract

Smart Contract

- Smart Contract는 Dapp개발자가 개발한 서비스 Logic을 네트워크에 등록하는 것이다.
- 사용자는 Contract를 실행하게 되면 그 Contract의 State를 변경하거나 조회한다.
- 사용자가 변경한 State는 블록체인상에 등록된 Contract Logic과 State에 따라 위변조없이 동작함을 보장한다.
- Turing complet 한 시스템이다.



개발 언어

Solidity

가장 많은 개발자들이 사용하는 언어로, C++ 기반의 언어이다. Object 기반 언어이며 상속, Library, 사용자 지정 type등을 지원한다.

Vyper

Python 기반 언어이다. 보안을 위해 Solidity에 비해 더 작은 기능을 가지고 있다.(modifiers, 상속, inline assembly, overload 미지원)

Yul & Yul+

초보자가 하기 어려운 언어이다. EVM 뿐 아니라 Ewasm도 지원한다. Yul+은 Low-Level언어로 Yul에 비해 더 효율적인 개발이 가능하다.

FE

가장 최근 개발된 언어로 2021년 1월 출시하였다. Ethereum에 처음 도전하는 개발하는 사용자를 위해 개발되었으며, Python과 Rust 기반 언어이다.

Solidity

- Contract는 오른쪽 화면과 같은 형식으로 되어있으며, 블록체인 Storage에 저장될 중요한 데이터와 이를 처리하는 함수로 구성되어있다.

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

contract Storage {

    uint256 number;

    function store(uint256 num) public {
        number = num;
    }

    function retrieve() public view returns (uint256){
        return number;
    }
}
```

Compile Contract

- Smart Contract를 Compile하게 되면 Bytecode, OPCODE, ABI를 얻게 된다. Contract를 네트워크에 배포할 때는 Bytecode가 필요하다. Contract를 실행할때 Gas 계산을 위해서는 OPCODE가 필요하다. ABI를 통해서 Client에서 정해진 Interface로 Contract와 통신을 할 수 있다.

Smart Contract

compile

Bytecode

```
608060405234801561001057600080fd5b5061015
080610020600396000f3fe6080604052348015
61001057600080fd5b50600436106100365760000
35
```

OPCODE

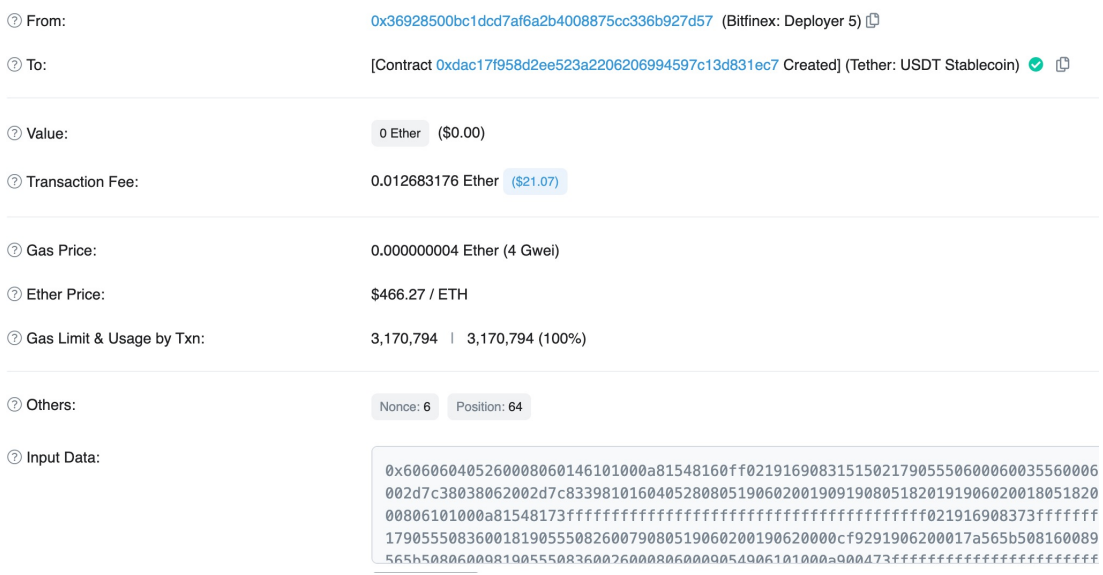
```
PUSH1 0x80 PUSH1 0x40 MSTORE CALLVALUE
DUP1 ISZERO PUSH2 0x10 JUMPI PUSH1 0x0 DUP1
REVERT JUMPDEST POP PUSH2 0x150 DUP1 PUSH2
0x20
```

ABI

```
[{"constant":true,"inputs":[],"name":"name","outputs":
[{"name":"","type":"string"}],"payable":false,"stateMutability":"
view","type":"function"},{"constant":false,"inputs":
[{"name":"_upgradedAddress","type":"address"}],"name":"deprecate"
,"outputs":
```

Contract Deploy

- Smart Contract Compile 후 생성된 Bytecode를 Ethereum Network 상에 배포하게 되면 Contract Address 생성과 함께 해당 주소에 Code가 블록체인상에 저장된다.



(출처 : etherscan.io)

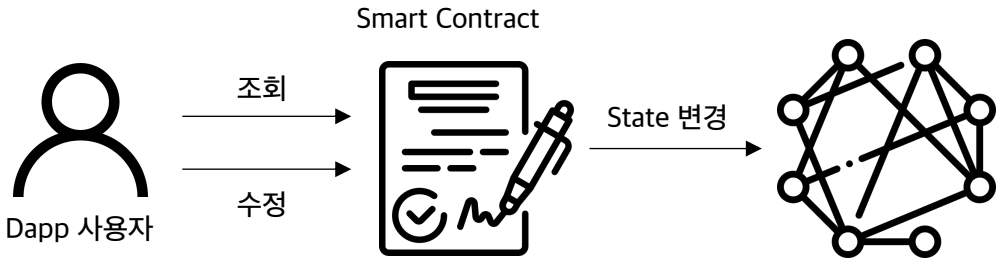
Contract 접근

- Smart Contract에 접근하기 위해서 필요한 것은 Contract Address와 ABI이다. 이를 통해서 Dapp 상에서 Smart Contract 코드 상의 함수를 호출할 수 있다.

```
>>> contract_id, contract_interface = compiled_sol.popitem()
>>> bytecode = contract_interface['bin']
>>> abi = contract_interface['abi']
>>> w3 = Web3(Web3.EthereumTesterProvider())
>>> Greeter = w3.eth.contract(abi=abi, bytecode=bytecode)
```

Contract 호출

- 사용자가 Contract 함수를 호출할 때 2가지 방식이 적용된다.
- 먼저 조회 함수를 호출하는 경우(state 변경이 없을 때) 이는 블록체인상에 조회 요청이 기록되지 않는다.
- 수정 함수를 호출하는 경우(state 변경이 있을 때) 블록체인상에 수정하는 호출에 대한 Transaction이 기록된다.



Contract 간 호출

- Contract간의 통신은 Internal Transaction으로 블록체인 상에 기록되지 않는다.
- trace_block을 통해서 조회가 가능하다. 하지만 가장 먼저 호출한 Dapp 사용자의 Transaction은 블록체인 상에 기록된다.
- delegate call, staticcall, call,transfer, selfdestruct 등의 함수를 이용해서 사용이 가능하다.
- Upgradeable(Proxy) Contract의 기본 개념이 된다.