한 번에 끝내는 블록체인, dApp 개발의 모든 것

Chapter 4

Lottery 컨트랙트 v2 개발

Chapter 4 Lottery 컨트랙트 v2 개발

Lottery 컨트랙트에 Chainlink VRF Consumer 구조 적용하기

Chainlink 컨트랙트 패키지 설치

- VRF 컨트랙트 사용을 위해, Chainlink 컨트랙트 패키지 설치
  - https://www.npmjs.com/package/@chainlink/contracts
- VRFCoordinatorV2Interface.sol, VRFConsumerV2Base.sol import
- LotteryV2 컨트랙트는 VRFConsumerBaseV2 컨트랙트 상속

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.15;

import "@chainlink/contracts/src/v0.8/interfaces/VRFCoordinatorV2Interface.sol";
import "@chainlink/contracts/src/v0.8/VRFConsumerBaseV2.sol";

contract LotteryV2 is VRFConsumerBaseV2 {
```

Chainlink VRF Consumer 구조 적용하기

- callbackGasLimit: 예시 10만 → 100만으로 늘려주기
  - callback에서 복잡한 작업해줄 것이므로 충분히 증가시켜야함
- numWords: 하나의 lottery 회차당 랜덤 값은 하나 필요하므로 1로 수정
- 생성자에서 VRF Coordinator 및 subscription id 세팅해줌

```
VRFCoordinatorV2Interface COORDINATOR;

// Your subscription ID.
uint64 s_subscriptionId;

// Goerli coordinator. For other networks,
// see https://docs.chain.link/docs/vrf-contracts/#configurations
address vrfCoordinator = 0x2Ca8E0C643bDe4C2E08ab1fA0da3401AdAD7734D;

// The gas lane to use, which specifies the maximum gas price to bump to.
// For a list of available gas lanes on each network,
// see https://docs.chain.link/docs/vrf-contracts/#configurations
bytes32 keyHash = 0x79d3d8832d904592c0bf9818b621522c988bb8b0c05cdc3b15aea1b6e8db0c15;

// Depends on the number of requested values that you want sent to the
// fulfillRandomWords() function. Storing each word costs about 20,000 gas,
// so 100,000 is a safe default for this example contract. Test and adjust
// this limit based on the network that you select, the size of the request,
// and the processing of the callback request in the fulfillRandomWords()
// function.
uint32 callbackGasLimit = 1000000;

// The default is 3, but you can set this higher.
uint16 requestConfirmations = 3;

// For this example, retrieve 2 random values in one request.
// Cannot exceed VRFCoordinatorV2.MAX_NUM_WORDS.
uint32 numWords = 1;

uint256[] public s_randomWords;
uint256 public s_requestId;

constructor(uint64 subscriptionId) VRFConsumerBaseV2(vrfCoordinator) {
    owner = msg.sender;

    COORDINATOR = VRFCoordinatorV2Interface(vrfCoordinator);
    s_subscriptionId = subscriptionId;
}
```

Chainlink VRF Consumer 구조 적용하기

- V1에서 pickWinner시 랜덤 값으로부터 winner 선정 후 바로 winner에게 상금 전송하던 구조
- Chainlink VRF에 랜덤 값 요청시 block confirmation 기간으로 인해 바로 winner를 선정할 수 없게됨
- 랜덤 값 요청 및 이로부터 선정된 winner에게 추후에 상금 전송하는 로직으로 분리 필요

```
function pickWinner() public onlyOwner {
    _requestRandomWords();
}

function _prizeWinner() internal {
    uint256 index = s_randomWords[0] % players.length;

    lotteryHistory[lotteryId] = players[index];
    lotteryId++;

    address payable winner = players[index];
    players = new address payable[](0);

    (bool success, ) = winner.call{value: address(this).balance}("");
    require(success, "Failed to send Ether");
}
```

Chainlink VRF Consumer 구조 적용하기

- pickWinner()에서 Chainlink VRF에 랜덤 값 요청하므로, _requestRandomWords()는 internal로 설정
- VRF Coordinator에서 랜덤 값 콜백으로 전송시 fulfillRandomWords() 실행됨
  - 이 함수에서 _prizeWinner() 호출하도록 설정
  - 랜덤 값 얻는 즉시 winner 선정 및 winner에게 상금 전송 이루어짐
  - 이 작업으로 인해 callbackGasLimit 상향 조정 필요

```
// Assumes the subscription is funded sufficiently.
function _requestRandomWords() internal {
    // Will revert if subscription is not set and funded.
    s_requestId = COORDINATOR.requestRandomWords(
        keyHash,
        s_subscriptionId,
        requestConfirmations,
        callbackGasLimit,
        numWords
    );
}

function fulfillRandomWords(
    uint256, /* requestId */
    uint256[] memory randomWords
) internal override {
    s_randomWords = randomWords;
    _prizeWinner();
}
```