Deploy an Upgradeable Contract

Umumnya contract yang sudah di deploy di blockchain sifatnya immutable, ngga bisa diubah. Di beberapa kondisi, kita perlu memperbarui logic di smart contract, seperti jika ada bug, vulnerability, menngikuti perkembangan market atau sekedar melakukan update fungisonalitas. Solusinya apa? Upgradeable contract. Dengan ini state data di contract terjaga dan fungsi bisa diubah. Modul ini akan menjelaskan konsep upgradeable contract sampai tahap deployment.

Contents

Prerequisites 3

Concept 3

Initialize and Reinitialize 3

ERC1967 3

Upgrade Methods 3

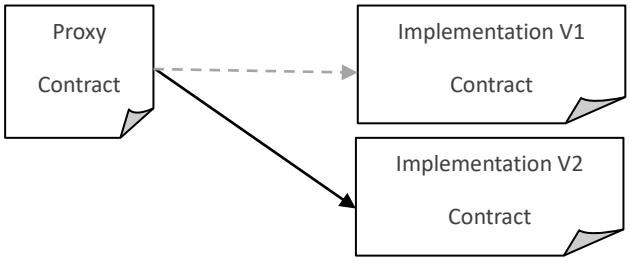
Contract Guide 4

Prerequisites

- Pemahaman tentang smart contract dasar
- Pemahaman tentang smart contract inheritance
- Pemahaman tentang Hardhat commands (optional, bisa via Remix online compiler)

Concept

Dalam upgradeable contract, kita akan deploy 2 contract, proxy contract dan execution/implemetation contract. Proxy contract menyimpan data state dan akan menunjuk execution contract untuk fungsi yang dapat dieksekusi terhadap data tersebut. Karena ini, architecture pattern-nya disebut "Proxy Pattern". Definisi proxy adalah bagian software yang punya kontrol terhadap suatu nagian lain dari software tersebut.



Dalam upgradeable smart contract, kita tidak mengenal constructor, yang kita gunakan adalah **initialize**. OpenZeppelin menghadirkan beberapa proxy patterns: Transparent, UUPS (Unified Upgradeable Proxy Standard), dan Beacon. UUPS lebih murah untuk penggunaan jangka panjang dan lebih terkenal, makan guide ini akan menggunakan UUPS. Biasa UUPS digunakan untuk smart contract wallet karena implementationnya cukup di-deploy satu kali tapi proxy contract bisa di-deploy bekali-kali menunjuk ke implementation contract yang sama.

Initialize and Reinitialize

Initialize hanya dipanggil satu kali saat implementation deployment pertama, jika ada state value yang perlu di-initialize pada implementation versi berikutnya, kita gunakan **reinitializer** function modifier dari **Initializable** library. Reinitializer ini punya 1 parameter untuk versi. Untuk penggunaan kedua, parameternya angka 2 karena angka 1 digunakan pada deployment pertama. Reinitializer perlu dipanggil secara **manual**, berbeda dengan initialize yang **otomatis saat deployment dengan Hardhat**. **Initialize perlu dilakukan manual jika deployment dengan metode lain yang tidak otomatis memanggil initialize()**. Reinitializable bersifat optional dan versi pada parameter hanya untuk menjaga tidak terpanggil lebih dari sekali.

ERC1967

Sebelum masuk ke contract guide, kita kenalan singkat dengan landasan dari upgradeable contract, ERC1967, standar terkait storage management untuk upgradeable contract. Dalam ERC1967, ada istilah proxy admin, address yang memiliki hak untuk mengubah implementation contract. UUPS menggunakan prinsip dari ERC1967, tapi mengganti proxy admin dengan **authorize upgrade** yang aksesnya perlu dijaga. Kalian bisa cek dokumentasi OpeZeppelin untuk mempelajari lebih dalam terkait proxy pattern dalam smart contract.

Upgrade Methods

Ada 2 cara yang bisa digunakan untuk upgrdae execution contract:

1. Inheritance: Contract versi baru inherit contract sebelumnya dan menambah atau mengubah function. Penggunaan modifier **virtual, override, dan internal** perlu diperhatikan jika menggunakan metode ini.
2. Rewrite: Memasukkan ulang seluruh function dan variable yang diperlukan ke contract versi baru. Perlu hati-hati biar ngga storage collision atau menyenggol existing logic.

Penggunaan variable/function version di contract atau version registry bisa membantu keep track contract version.

Feel free to try it yourself before reading the guide.

Contract Guide

Setelah bikin project Hardhat baru, install node modules yang diperlukan dan buat 2 contract untuk implementasi V1 dan V2. Di sini kita akan menggunakan inheritance method.

1. Berikut isi package.json. Gunakan **npm i –legacy-peer-deps** kalau instalasi bermasalah. Kita pakai **@openzeppelin/contracts-upgradeable** untuk library khusus upgradeable contract dan **@openzeppelin/hardhat-upgrades** untuk deployment dan function berkaitan dengan upgradeable contract.

```
{
  "name": "hardhat-project",
  "dependencies": {
    "@nomicfoundation/hardhat-toolbox": "^3.0.0",
    "@openzeppelin/contracts-upgradeable": "^4.9.6"
  },
  "devDependencies": {
    "@nomicfoundation/hardhat-chai-matchers": "^2.0.6",
    "@nomicfoundation/hardhat-ethers": "^3.0.6",
    "@openzeppelin/hardhat-upgrades": "^3.1.0",
    "chai": "^4.3.7",
    "dotenv": "^16.4.5",
    "hardhat": "^2.12.6",
    "ethers": "^6.12.1"
  }
}
```

2. Untuk contoh simple, kita akan bikin mailbox contract. Berikut MailboxV1. Yang wajib di import adalah **Initializable** dan **UUPSUpgradeable** untuk initializer function dan proxy pattern yang digunakan. Di sini ada **OwnableUpgradeable**, versi upgradeable dari library Ownable. Ini agar kita bisa pakai **onlyOwner** sebagai access control untuk authorize upgrade. Ini penting agar tidak sembarang orang dapat mengubah implementation contract.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";
import "@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol";
import "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";

contract MailboxV1 is Initializable, OwnableUpgradeable, UUPSUpgradeable {
    uint256 public version;
    mapping(address => string) internal messages;

    event MessageSent(address indexed sender, string message);
    event MessageReceived(address indexed recipient, string message);
```

```
    function initialize() public initializer {
        __Ownable_init();
        version = 1;
    }

    function sendMessage(address recipient, string memory message) public {
        require(recipient != address(0), "Invalid recipient");
        messages[recipient] = message;
        emit MessageSent(msg.sender, message);
    }

    function getMessage() public view returns (string memory) {
        return messages[msg.sender];
    }

    function _authorizeUpgrade(
        address newImplementation
    ) internal override onlyOwner {}
}
```

3. Berikut contract versi 2 yang inherit versi 1.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "./MailboxV1.sol";

contract MailboxV2 is MailboxV1 {
    function reinitialize() public reinitializer(2) {
        version = 2;
    }

    function forwardMessage(address recipient) public {
        require(
            bytes(messages[msg.sender]).length > 0,
            "No message to forward"
        );
        string memory message = messages[msg.sender];
        // delete messages[msg.sender]; // uncomment if you want to delete the message after
forwarding
        messages[recipient] = message;
        emit MessageReceived(recipient, message);
    }
}
```

4. Untuk deployment, function yang dipanggil adalah **deployProxy** dari hardhat-upgrades. Di sini, **getAddress()** akan memberi address proxy contract dan **getImplementationAddress()** akan memberi address implementation contract.

```
const MailboxFactory = await ethers.getContractFactory("MailboxV1");
mailboxProxy = await upgrades.deployProxy(MailboxFactory);

mailboxProxyAddress = await mailboxProxy.getAddress();
implementationAddress = await upgrades.erc1967.getImplementationAddress(mailboxProxyAddress);
```

5. Untuk mengubah implementation contract, kita panggil **upgradeProxy()** dengan parameter proxy address dan implementation contract factory yang baru. Jangan lupa **reinitialize**.

```
const MailboxV2Factory = await ethers.getContractFactory("MailboxV2");
let upgrade = await upgrades.upgradeProxy(mailboxProxyAddress, MailboxV2Factory);
await upgrade.reinitialize();
```

6. Run testing. Code lengkap bisa cek di Github.