

TECHNICAL REPORT DOCUMENTATION

Learn Blockchain, Solidity, and Full Stack Web3 Development with JavaScript



TELKOM UNIVERSITY

Hamzah Fatihulhaq - 1103191293
Delatifa Putri Sugandi - 1103194080

Lesson 1 : Blockchain Basics

- **Blockchain**

Blockchain adalah database atau buku besar terdistribusi yang dibagikan di antara node jaringan komputer. Sebagai database, blockchain menyimpan informasi secara elektronik dalam format digital. Inovasi dengan blockchain adalah menjamin kesetiaan dan keamanan catatan data dan menghasilkan kepercayaan tanpa perlu pihak ketiga yang terpercaya. Berbagai jenis informasi dapat disimpan di blockchain, tetapi penggunaan yang paling umum sejauh ini adalah sebagai buku besar untuk transaksi. Dalam kasus Bitcoin, blockchain digunakan dengan cara yang terdesentralisasi sehingga tidak ada satu orang atau kelompok yang memiliki kendali sebaliknya, semua pengguna secara kolektif mempertahankan kendali. Blockchain terdesentralisasi tidak dapat diubah, yang berarti bahwa data yang dimasukkan tidak dapat diubah. Untuk Bitcoin, ini berarti bahwa transaksi dicatat secara permanen dan dapat dilihat oleh siapa saja.

- **Smart Contract**

Smart Contract adalah program komputer yang di-host dan dieksekusi di jaringan blockchain. Setiap kontrak cerdas terdiri dari kode yang menentukan kondisi yang telah ditentukan sebelumnya, yang bila dipenuhi, memicu hasil. Dengan berjalan di blockchain terdesentralisasi alih-alih server terpusat, kontrak pintar memungkinkan banyak pihak untuk mencapai hasil bersama dengan cara yang akurat, tepat waktu, dan anti-rusak. Smart Contract juga merupakan infrastruktur yang kuat untuk otomatisasi karena tidak dikendalikan oleh administrator pusat dan tidak rentan terhadap satu titik serangan oleh entitas jahat. Ketika diterapkan pada perjanjian digital multi-pihak, aplikasi kontrak pintar dapat mengurangi risiko pihak lawan, meningkatkan efisiensi, menurunkan biaya, dan memberikan tingkat transparansi baru ke dalam proses.

- **Ethereum**

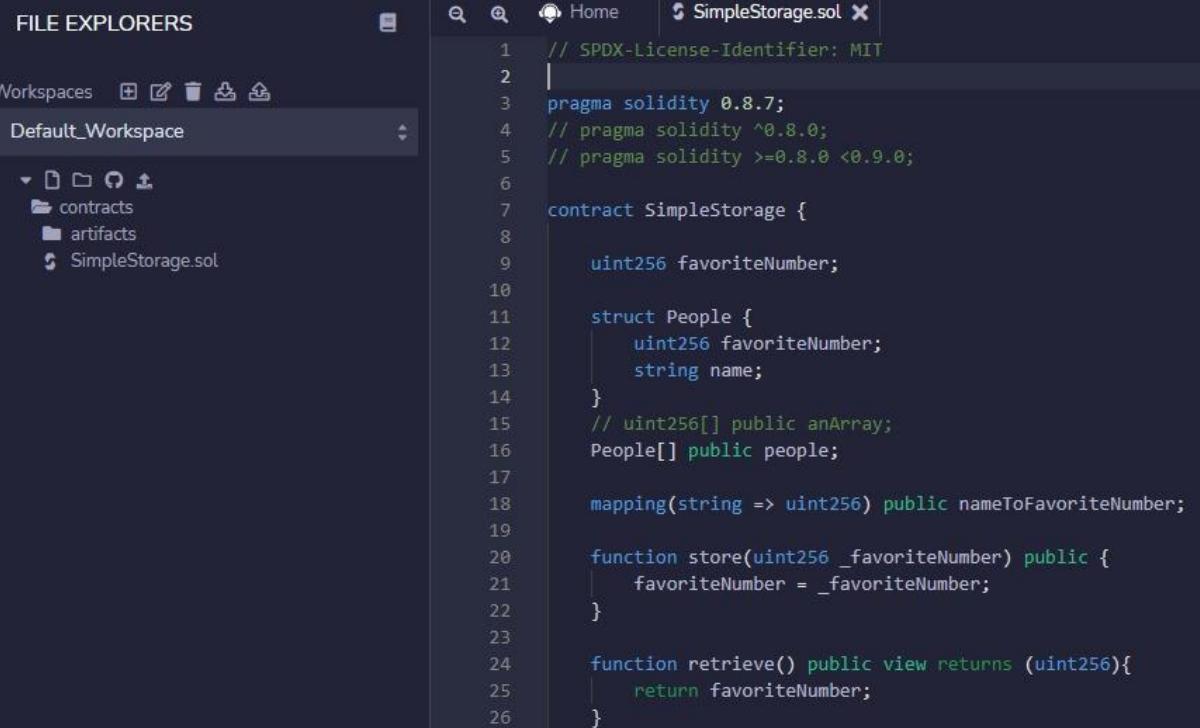
Ethereum adalah untuk membuat protokol alternatif untuk membangun aplikasi terdesentralisasi, memberikan rangkaian pengorbanan berbeda yang kami yakini akan sangat berguna untuk kelas besar aplikasi terdesentralisasi, dengan penekanan khusus pada situasi di mana waktu pengembangan yang cepat, keamanan untuk aplikasi kecil dan aplikasi yang jarang digunakan, dan kemampuan aplikasi yang berbeda untuk berinteraksi dengan sangat efisien, adalah penting.

- **Decentralized Oracle**

Oracle terdesentralisasi adalah sekelompok oracle blockchain independen yang menyediakan data ke blockchain . Setiap node atau oracle independen dalam jaringan oracle terdesentralisasi secara independen mengambil data dari sumber di luar rantai dan membawanya ke dalam rantai. Data tersebut kemudian dikumpulkan sehingga sistem dapat mencapai nilai kebenaran deterministik untuk titik data tersebut. Oracle terdesentralisasi memecahkan masalah oracle.

Lesson 2: Welcome to Remix! Simple Storage

Hal pertama yang harus dilakukan adalah membuat contract pada <https://remix.ethereum.org>.



The screenshot shows the Remix IDE interface. On the left, there's a 'FILE EXPLORERS' sidebar with a 'Default_Workspace' section containing a 'contracts' folder, an 'artifacts' folder, and a file named 'SimpleStorage.sol'. The main area is titled 'SimpleStorage.sol' and contains the Solidity code for the Simple Storage contract:

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.7;
// pragma solidity ^0.8.0;
// pragma solidity >=0.8.0 <0.9.0;

contract SimpleStorage {
    uint256 favoriteNumber;

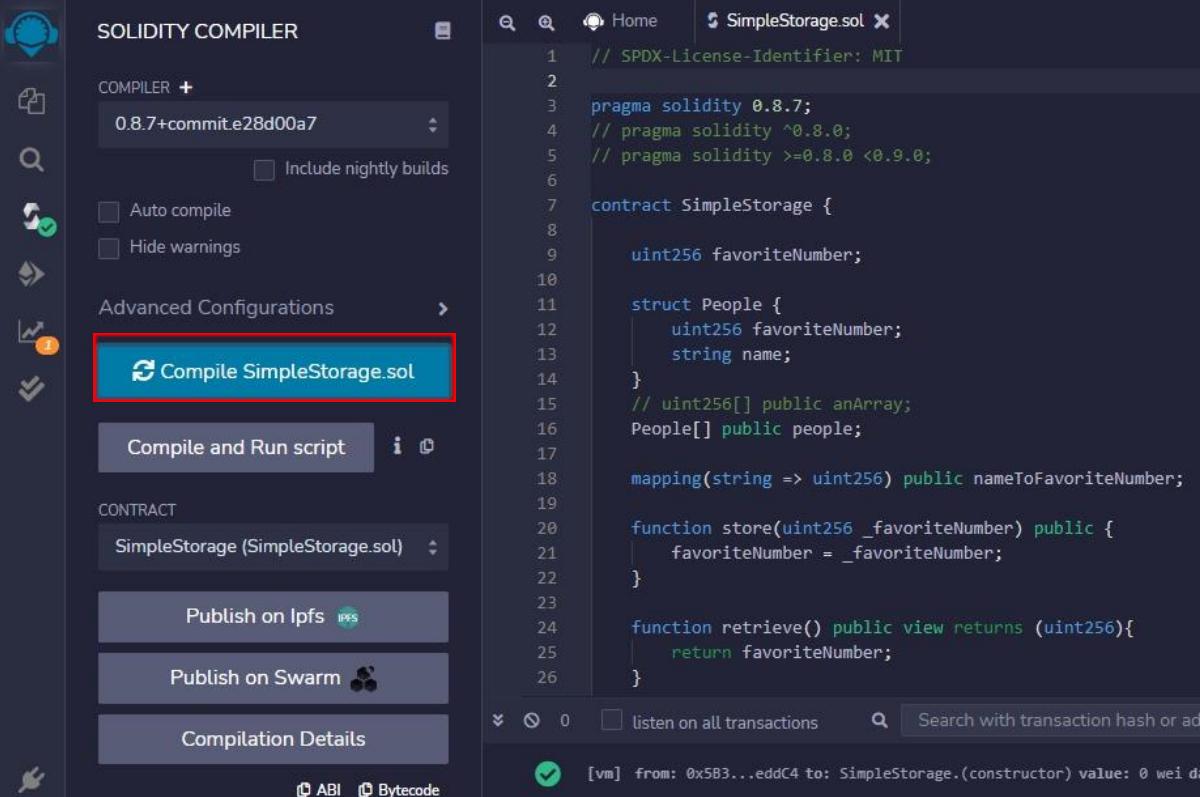
    struct People {
        uint256 favoriteNumber;
        string name;
    }
    // uint256[] public anArray;
    People[] public people;

    mapping(string => uint256) public nameToFavoriteNumber;

    function store(uint256 _favoriteNumber) public {
        favoriteNumber = _favoriteNumber;
    }

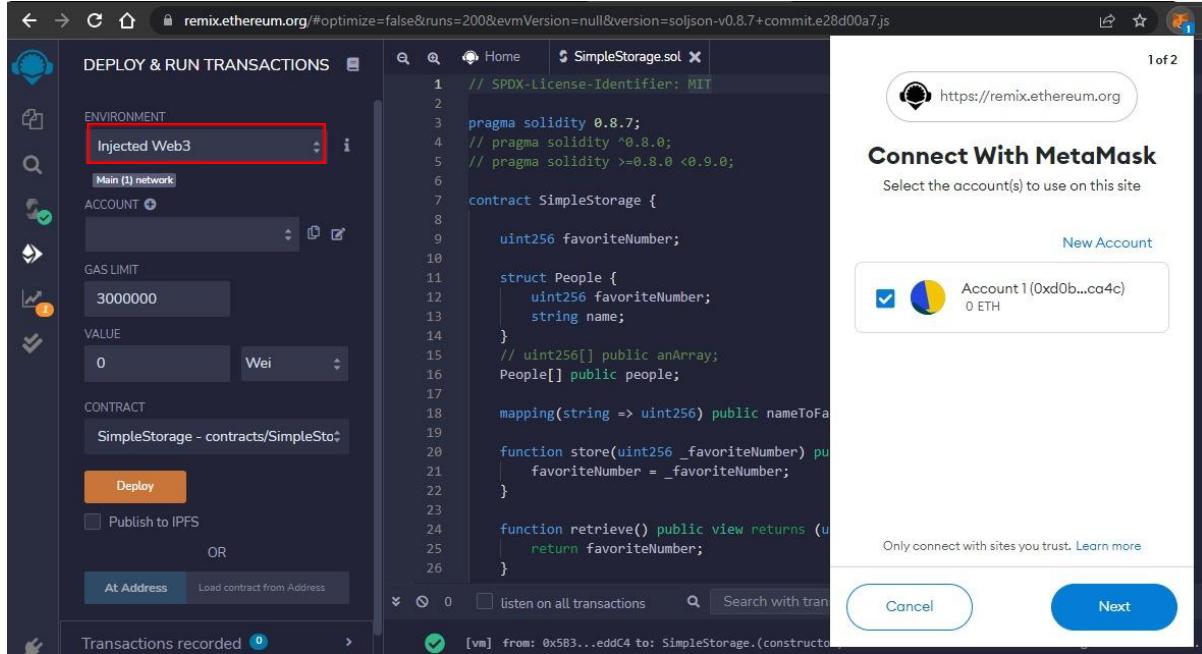
    function retrieve() public view returns (uint256){
        return favoriteNumber;
    }
}
```

Setelah SimpleStorage.sol sudah tertulis, klik tombol compile.



The screenshot shows the Solidity Compiler interface within the Remix IDE. On the left, there's a sidebar with various icons and settings like 'Auto compile' and 'Hide warnings'. The main area is titled 'SimpleStorage.sol' and contains the same Solidity code as before. A red box highlights the 'Compile SimpleStorage.sol' button, which is located in the 'Advanced Configurations' section. Below it are buttons for 'Compile and Run script', 'Publish on Ipfs', 'Publish on Swarm', and 'Compilation Details'.

Sebelum proses deploy, Gunakan Environment Injected Web3, kemudian sambungkan wallet metamask kita terlebih dahulu. Dan pilih Account yang sesuai.



The screenshot shows the Remix IDE interface. On the left, there's a sidebar with various icons for file operations, deployment, and monitoring. The main area has tabs for 'Home' and 'SimpleStorage.sol'. The code for SimpleStorage.sol is pasted below:

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.7;
// pragma solidity ^0.8.0;
// pragma solidity >=0.8.0 <0.9.0;

contract SimpleStorage {
    uint256 favoriteNumber;
    struct People {
        uint256 favoriteNumber;
        string name;
    }
    // uint256[] public anArray;
    People[] public people;

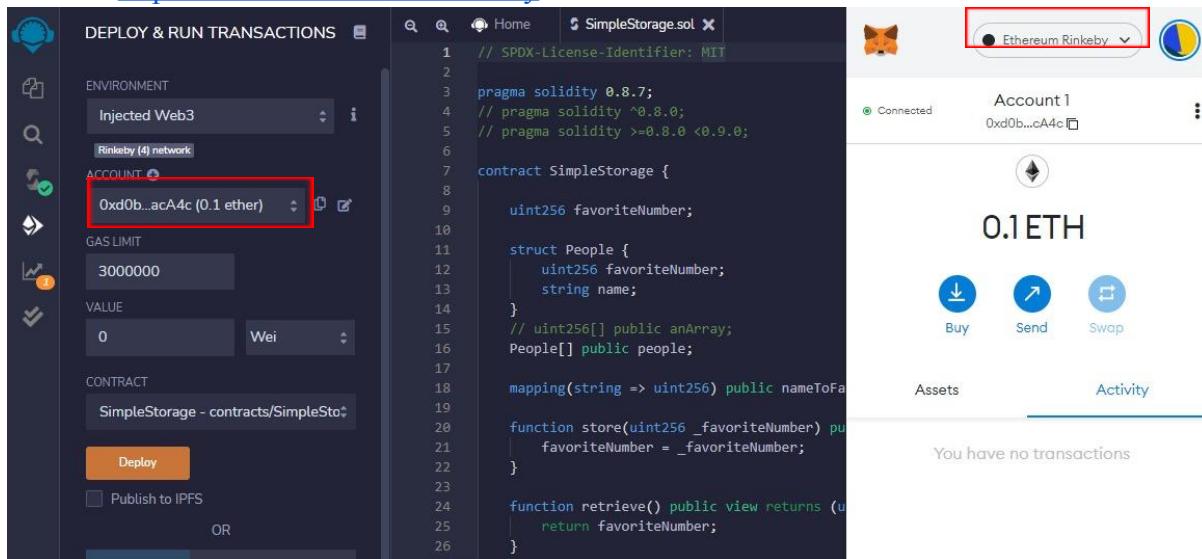
    mapping(string => uint256) public nameToFavoriteNumber;

    function store(uint256 _favoriteNumber) public {
        favoriteNumber = _favoriteNumber;
    }

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

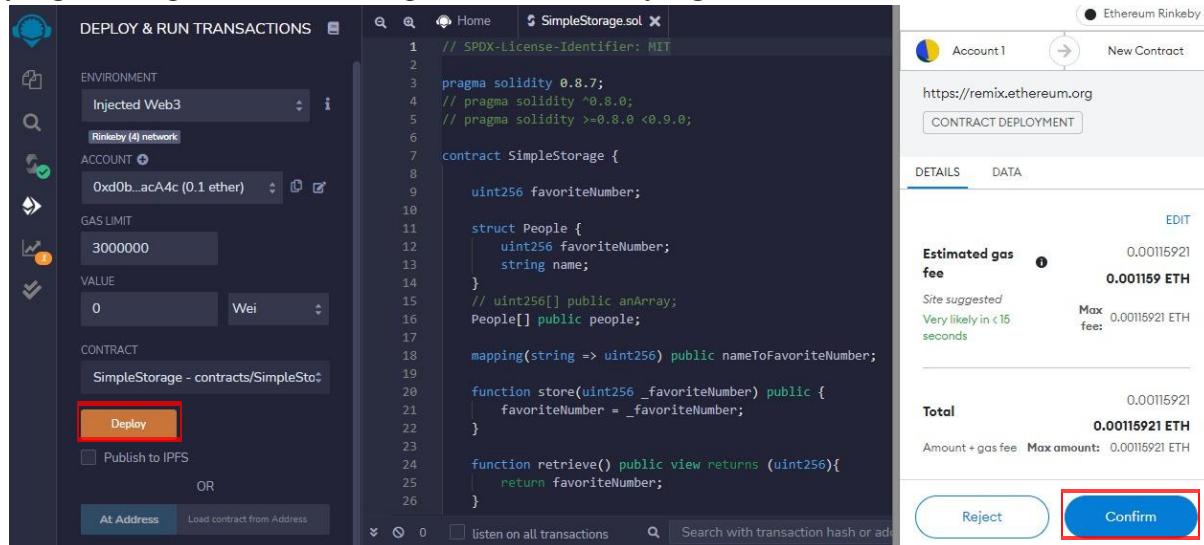
Below the code, there are buttons for 'Deploy', 'Publish to IPFS', and 'At Address'. The 'Deploy' button is highlighted. To the right, a MetaMask connection dialog is displayed, showing 'Connect With MetaMask' and 'Select the account(s) to use on this site'. It lists 'Account 1 (0xd0b...ca4c) 0 ETH' with a checked checkbox. At the bottom right of the dialog are 'Cancel' and 'Next' buttons.

Pada contoh ini kami akan melakukan Deploy dengan menggunakan Network Ethereum Rinkeby. Untuk menggunakan Ethereum Rinkeby sebagai Testnet dapat diakses dari link berikut : <https://faucets.chain.link/rinkeby>



The screenshot shows the Remix IDE interface with the environment set to 'Ethereum Rinkeby'. The MetaMask connection dialog is still present, showing 'Connected' status for 'Account 1 (0xd0b...ca4c) 0.1 ETH'. The 'Assets' tab is active, displaying '0.1 ETH' and three buttons: 'Buy', 'Send', and 'Swap'. Below the balance, it says 'You have no transactions'. The code for SimpleStorage.sol remains the same as in the previous screenshot.

Ketika tombol deploy diklik maka akan muncul pop up Metamask seperti gambar dibawah yang menampilkan details dari proses transaction yang akan dilakukan.



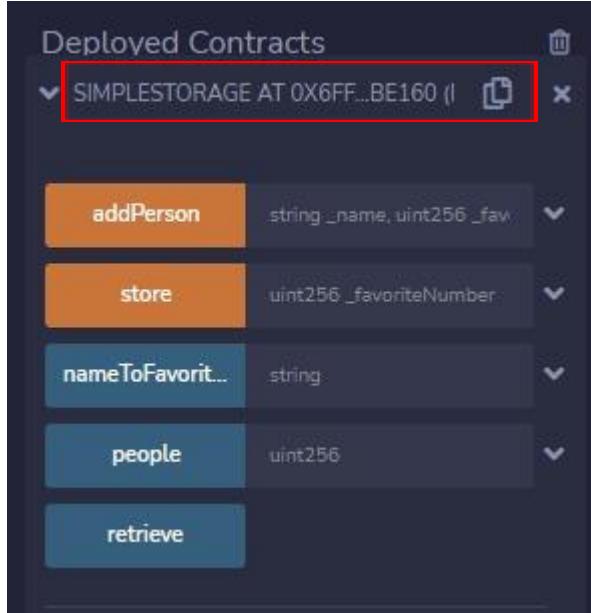
Setelah mengkonfirmas transaksi maka akan muncul details proses seperti dibawah. Setelah proses sudah selesai, dapat dilakukan klik view on etherscan.



Setelah diklik, akan menuju ke <https://rinkeby.etherscan.io/> dan terlihat dengan jelas seluruh details dari transaksi yang telah berhasil.

Detail	Value
From:	0xd0bce4f704c8955cd5e9f7d78d5fe57b1beaca4c
To:	[Contract 0x6ff975030503b3ce7e2c1db625c79a214bdbe160 Created]
Value:	0 Ether (\$0.00)
Transaction Fee:	0.0001159205004173138 Ether (\$0.00)
Gas Price:	0.000000002500000009 Ether (2.500000009 Gwei)

Dan jika kita kembali ke Contracts SimpleStorage kita yang terdapat di Remix maka akan tampil seperti gambar dibawah pada bagian Deploy Transaction. Ini menandakan proses deploy telah selesai dan kita juga dapat melakukan pengecekan kembali pada <https://rinkeby.etherscan.io> mengenai transaksi yang telah berlangsung dengan mengcopy contracts tersebut.



Paste pada bar pencarian, dan akan ditemukan transaksi contracts yang dicari.

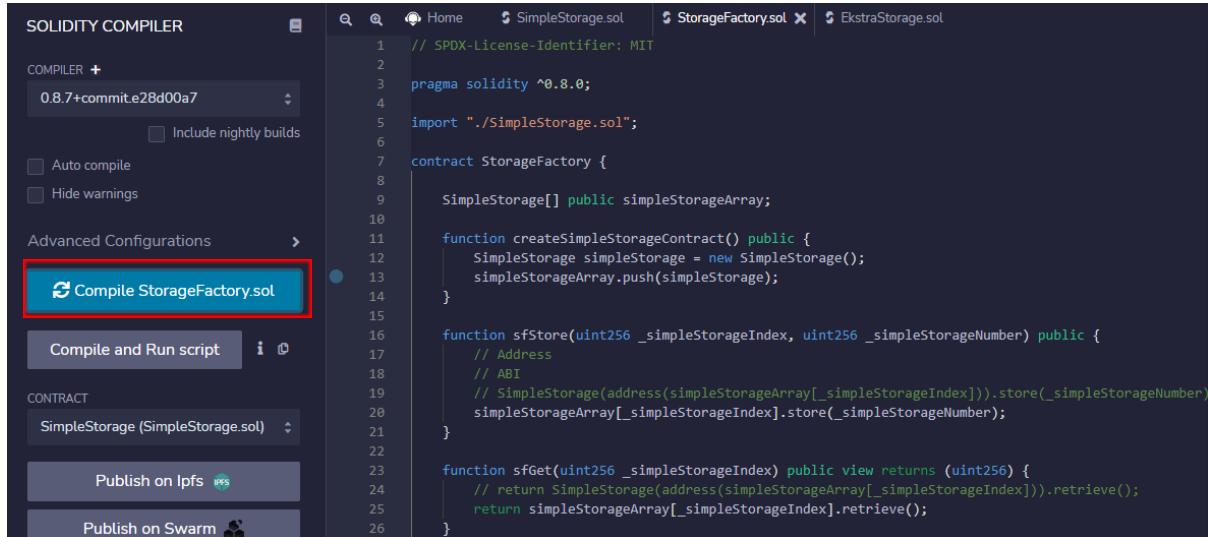
The screenshot shows the Etherscan interface for the Rinkeby Testnet Network. It displays the 'Contract Overview' for the contract at address 0x6ff975030503B3ce7e2C1DB625C79A214BdBe160. The balance is 0 Ether. The 'Transactions' tab is selected, showing one transaction: 0x7d3703354fca89aa9c... (Method: Contract Creation, Value: 0 Ether, Txn Fee: 0.0011592). The transaction was created by 0xd0bce4f704c8955cd5e... at block 10948530, 16 mins ago.

Lesson 3: Remix Storage Factory

Hal yang paling utama harus dibuat adalah membuat beberapa file tersebut didalam Remix.

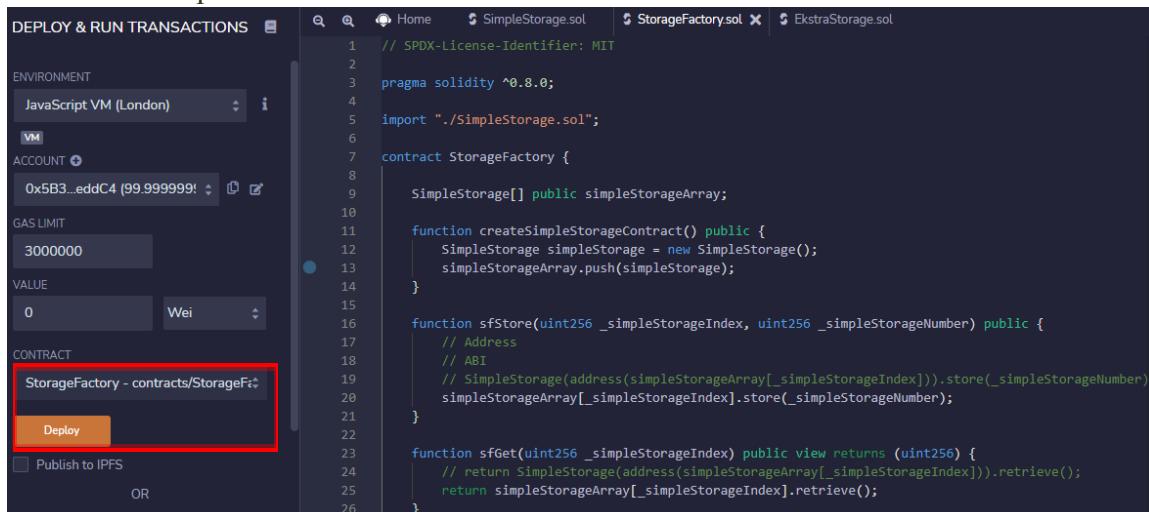
- Simple Storage.sol
- StorageFactory.sol
- EkstraStorage.sol

Setelah ketiga file tersebut dibuat. Lakukan Compile dan deploy untuk file StorageFactory.sol



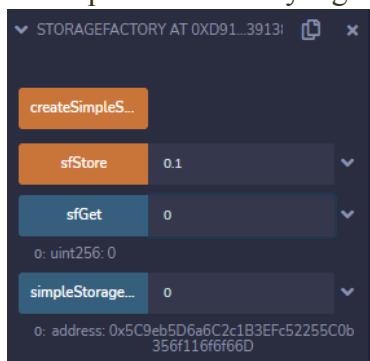
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
import "./SimpleStorage.sol";
contract StorageFactory {
    SimpleStorage[] public simpleStorageArray;
    function createSimpleStorageContract() public {
        SimpleStorage simpleStorage = new SimpleStorage();
        simpleStorageArray.push(simpleStorage);
    }
    function sfStore(uint256 _simpleStorageIndex, uint256 _simpleStorageNumber) public {
        // Address
        // ABI
        // SimpleStorage(address(simpleStorageArray[_simpleStorageIndex])).store(_simpleStorageNumber)
        simpleStorageArray[_simpleStorageIndex].store(_simpleStorageNumber);
    }
    function sfGet(uint256 _simpleStorageIndex) public view returns (uint256) {
        // return SimpleStorage(address(simpleStorageArray[_simpleStorageIndex])).retrieve();
        return simpleStorageArray[_simpleStorageIndex].retrieve();
    }
}
```

Sebelum klik tombol deploy, jangan lupa untuk memilih contract yang sesuai dengan file sudah kita compile.



```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
import "./SimpleStorage.sol";
contract StorageFactory {
    SimpleStorage[] public simpleStorageArray;
    function createSimpleStorageContract() public {
        SimpleStorage simpleStorage = new SimpleStorage();
        simpleStorageArray.push(simpleStorage);
    }
    function sfStore(uint256 _simpleStorageIndex, uint256 _simpleStorageNumber) public {
        // Address
        // ABI
        // SimpleStorage(address(simpleStorageArray[_simpleStorageIndex])).store(_simpleStorageNumber)
        simpleStorageArray[_simpleStorageIndex].store(_simpleStorageNumber);
    }
    function sfGet(uint256 _simpleStorageIndex) public view returns (uint256) {
        // return SimpleStorage(address(simpleStorageArray[_simpleStorageIndex])).retrieve();
        return simpleStorageArray[_simpleStorageIndex].retrieve();
    }
}
```

Lakukan Crate Transaksi seperti gambar dibawah. Jika sudah selesai maka kita akan mendapatkan address yang dapat di inputkan ke simpleStorage array.



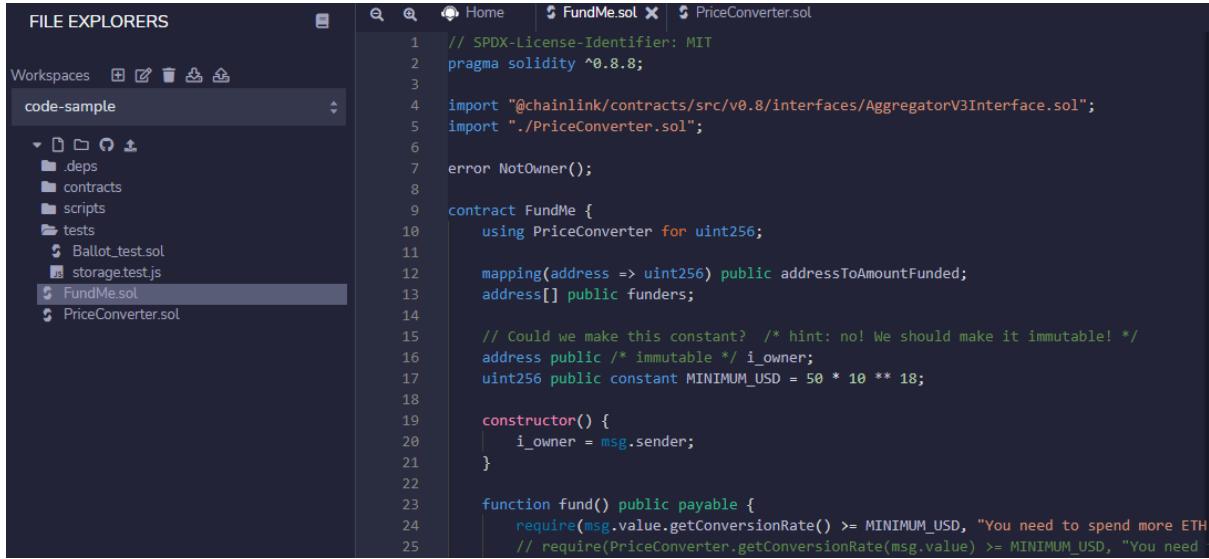
STORAGEFACTORY AT 0xD91...3913:

createSimpleS...	
sfStore	0.1
sfGet	0
0: uint256: 0	
simpleStorage...	0
0: address: 0x5C9eb5D6a6C2c1B3EfC52255C0b356f116ff66D	

Lesson 4: Remix Fund Me

Kunjungi <https://remix.ethereum.org/> dan buat beberapa file berikut:

1. FundMe.sol



```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.8;

import "@chainlink/contracts/src/v0.8/interfaces/AggregatorV3Interface.sol";
import "./PriceConverter.sol";

error NotOwner();

contract FundMe {
    using PriceConverter for uint256;

    mapping(address => uint256) public addressToAmountFunded;
    address[] public funders;

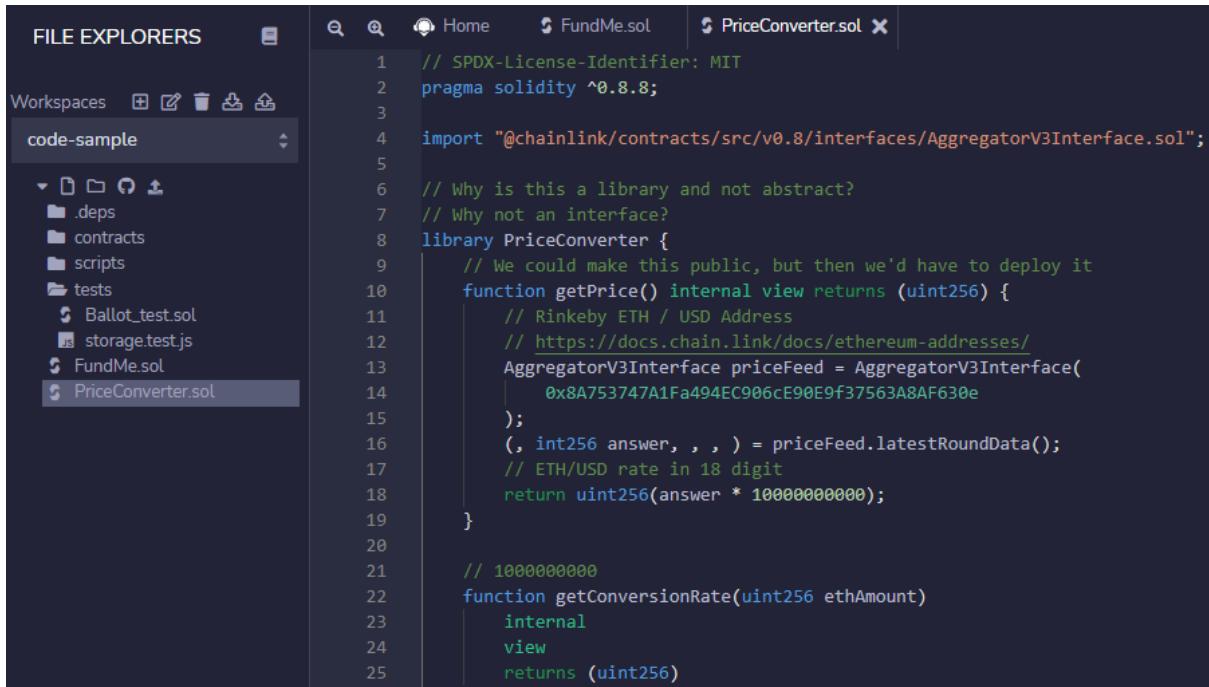
    // Could we make this constant? /* hint: no! We should make it immutable! */
    address public /* immutable */ i_owner;
    uint256 public constant MINIMUM_USD = 50 * 10 ** 18;

    constructor() {
        i_owner = msg.sender;
    }

    function fund() public payable {
        require(msg.value.getConversionRate() >= MINIMUM_USD, "You need to spend more ETH");
        // require(PriceConverter.getConversionRate(msg.value) >= MINIMUM_USD, "You need to spend more ETH");
    }
}
```

Fungsi FundMe.sol merupakan sebuah program etherium yang dapat memberikan fund dalam membayarkan antara transaksi etherium, ke sebuah mata uang seperti USD maupun dalam jenis Ethereum lainnya.

2. PriceConverter.sol



```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.8;

import "@chainlink/contracts/src/v0.8/interfaces/AggregatorV3Interface.sol";

// Why is this a library and not abstract?
// Why not an interface?
library PriceConverter {
    // We could make this public, but then we'd have to deploy it
    function getPrice() internal view returns (uint256) {
        // Rinkeby ETH / USD Address
        // https://docs.chain.link/docs/ethereum-addresses/
        AggregatorV3Interface priceFeed = AggregatorV3Interface(
            0x8A753747A1Fa494EC906cE90E9f37563A8AF630e
        );
        (, int256 answer, , , ) = priceFeed.latestRoundData();
        // ETH/USD rate in 18 digit
        return uint256(answer * 1000000000);
    }

    // 1000000000
    function getConversionRate(uint256 ethAmount)
        internal
        view
        returns (uint256)
    }
```

Mengirim ETH Melalui Fungsi & Pengembalian

Transaksi Ethereum mengacu pada tindakan yang diprakarsai oleh akun yang dimiliki secara eksternal, dengan kata lain akun yang dikelola oleh manusia, bukan kontrak. Misalnya, jika Bob mengirim Alice 1 ETH, akun Bob harus didebit dan akun Alice harus dikredit. Tindakan perubahan status ini terjadi dalam suatu transaksi.

Transaksi yang dikirimkan mencakup informasi berikut:

- recipient– alamat penerima (jika akun milik eksternal, transaksi akan mentransfer nilai. Jika akun kontrak, transaksi akan mengeksekusi kode kontrak)
 - signature– pengenal pengirim. Ini dihasilkan ketika kunci pribadi pengirim menandatangani transaksi dan mengonfirmasi bahwa pengirim telah mengotorisasi transaksi ini
 - value– jumlah ETH untuk ditransfer dari pengirim ke penerima (dalam WEI, denominasi ETH)
 - data– bidang opsional untuk memasukkan data arbitrer
 - gasLimit– jumlah maksimum unit gas yang dapat dikonsumsi oleh transaksi. Satuan gas mewakili langkah-langkah komputasi
 - maxPriorityFeePerGas– jumlah maksimum gas yang akan dimasukkan sebagai tip untuk penambang
 - maxFeePerGas– jumlah maksimum gas yang bersedia dibayarkan untuk transaksi (termasuk baseFeePerGas dan maxPriorityFeePerGas)

Sebagai salah satu contoh penggunaan.

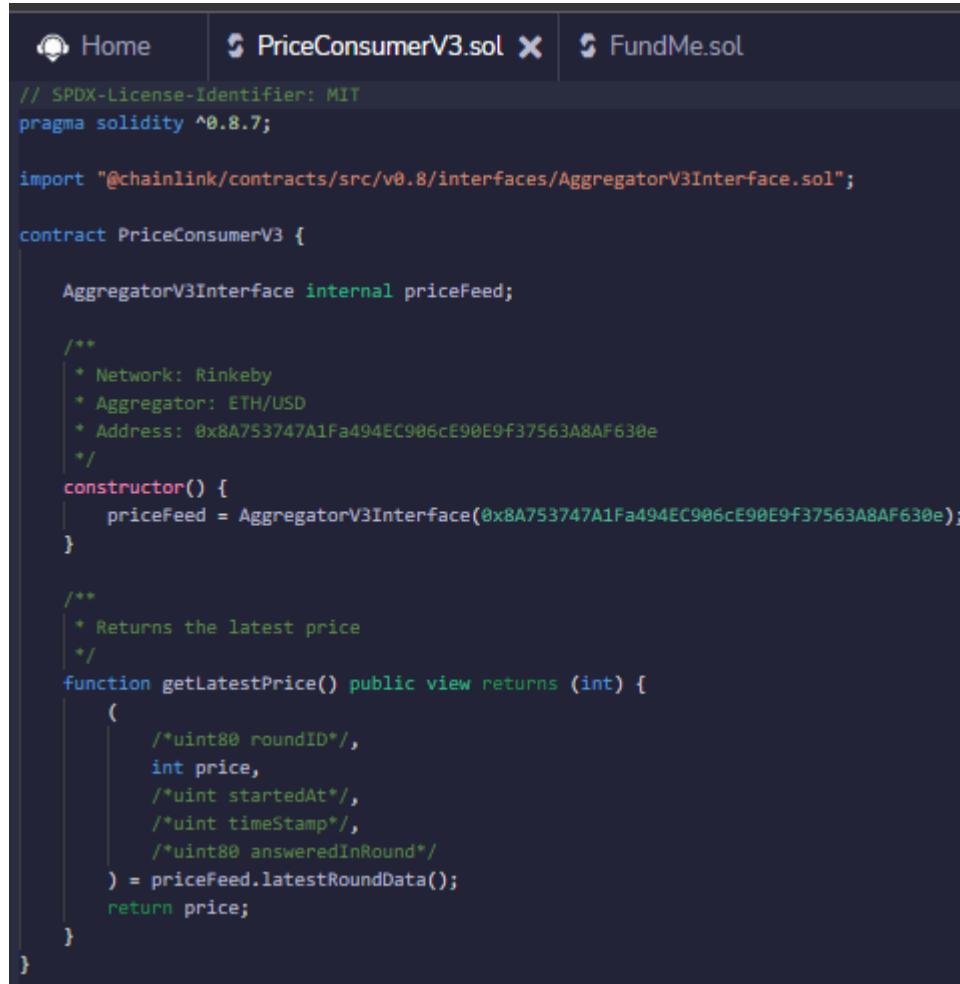
Menggunakan Data Feeds

Umpam Data Chainlink adalah cara tercepat untuk menghubungkan kontrak pintar Anda dengan harga pasar aset dunia nyata. Misalnya, salah satu kegunaan umpan data adalah mengaktifkan kontrak pintar untuk mengambil data harga terbaru dari suatu aset dalam satu panggilan.

Panduan ini berlaku khusus untuk menggunakan data feed pada EVM Chains. Untuk mendapatkan daftar lengkap Umpam Data Chainlink yang berjalan di Rantai EVM.

Solidity

Untuk menggunakan data harga, kontrak pintar Anda harus merujuk AggregatorV3Interface, yang mendefinisikan fungsi eksternal yang diterapkan oleh Umpam Data. Fungsi latestRoundData mengembalikan lima nilai yang mewakili informasi tentang data harga terbaru.



The screenshot shows a code editor interface with three tabs at the top: "Home", "PriceConsumerV3.sol", and "FundMe.sol". The "PriceConsumerV3.sol" tab is active, displaying the following Solidity code:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.7;

import "@chainlink/contracts/src/v0.8/interfaces/AggregatorV3Interface.sol";

contract PriceConsumerV3 {

    AggregatorV3Interface internal priceFeed;

    /**
     * Network: Rinkeby
     * Aggregator: ETH/USD
     * Address: 0x8A753747A1Fa494EC906cE90E9f37563A8AF630e
     */
    constructor() {
        priceFeed = AggregatorV3Interface(0x8A753747A1Fa494EC906cE90E9f37563A8AF630e);
    }

    /**
     * Returns the latest price
     */
    function getLatestPrice() public view returns (int) {
        (
            /*uint80 roundID*/,
            int price,
            /*uint startedAt*/,
            /*uint timeStamp*/,
            /*uint80 answeredInRound*/
        ) = priceFeed.latestRoundData();
        return price;
    }
}
```

Mengirim Eter (transfer, kirim, panggil)

Bagaimana cara mengirim Eter?

Anda dapat mengirim Eter ke kontrak lain dengan

- `transfer(2300 gas, melempar kesalahan)`
- `send(2300 gas, mengembalikan bool)`
- `call(majukan semua gas atau setel gas, kembalikan bool)`

Bagaimana cara menerima Eter?

Kontrak yang menerima Eter harus memiliki setidaknya salah satu fungsi di bawah ini

- `receive() external payable`
- `fallback() external payable`

`receive()` disebut jika `msg.data` kosong, jika `fallback()` tidak disebut.

Metode mana yang harus Anda gunakan?

`call` dikombinasikan dengan re-entrancy guard adalah metode yang direkomendasikan untuk digunakan setelah Desember 2019.

Jaga agar tidak masuk kembali dengan

- membuat semua perubahan status sebelum memanggil kontrak lain
- menggunakan pengubah penjaga masuk kembali

The screenshot shows a code editor with two tabs: 'ReceivEther.sol' and 'SendEther.sol'. The 'ReceivEther.sol' tab contains the following code:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.13;

contract ReceiveEther {
    /*
    Which function is called, fallback() or receive()?
    |
    | send Ether
    |   |
    |   msg.data is empty?
    |   / \
    |   yes  no
    |
    | receive() exists?  fallback()
    |   / \
    |   yes  no
    |   /
    |   receive()  fallback()
    */

    // Function to receive Ether. msg.data must be empty
    receive() external payable {}

    // Fallback function is called when msg.data is not empty
    fallback() external payable {}

    function getBalance() public view returns (uint) {
        return address(this).balance;
    }
}

contract SendEther {
    function sendViaTransfer(address payable _to) public payable {
        // This function is no longer recommended for sending Ether.
        _to.transfer(msg.value);
    }

    function sendViaSend(address payable _to) public payable {
        // Send returns a boolean value indicating success or failure.
        // This function is not recommended for sending Ether.
        bool sent = _to.send(msg.value);
        require(sent, "Failed to send Ether");
    }

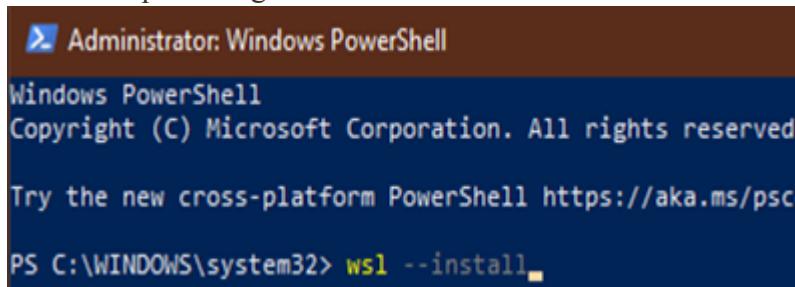
    function sendViaCall(address payable _to) public payable {
        // Call returns a boolean value indicating success or failure.
        // This is the current recommended method to use.
        (bool sent, bytes memory data) = _to.call{value: msg.value}("");
        require(sent, "Failed to send Ether");
    }
}
```

Lesson 5: Ethers.js Simple Storage

Dalam melakukan penyimpanan ether dibutuhkan beberapa tools diantaranya visual studio code, wsl, dan sebagainya.

Pada hal ini , Laptop yang saya gunakan menggunakan sistem operasi Windows. sehingga dapat dilakukan pengaturan seperti berikut:

- Windows Setup
- WSL
- When working in WSL, use Linux commands instead of Windows commands
- TroubleShooting
- curl -o https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.1/install.sh | bash



A screenshot of an Administrator Windows PowerShell window. The title bar says "Administrator: Windows PowerShell". The content shows the standard PowerShell welcome screen with copyright information. At the bottom, the command "PS C:\WINDOWS\system32> wsl --install" is visible, with the "wsl" part highlighted in green.

Step by step Persiapan :

1. Kloning repository berikut
git clone <https://github.com/PatrickAlphaC/ethers-simple-storage>
cd ethers-simple-storage
2. Kemudian instal dependensi
yarn
3. Jika Anda suka typescript, jalankan git checkout typescript lalu jalankan yarn
4. Jalankan yarn add ganache
5. Ubah pengaturan Server di Ganache
Pengaturan > Server > Nama Host
Ubah Nama Host menjadi vEthernet (WSL)
6. Jalankan aplikasi Anda
node deploy.js
7. Dilanjutkan dengan Menyebarluaskan ke testnet
Pastikan Anda memiliki metamask atau dompet lain, dan ekspor kunci pribadi.
8. Ekspor kunci pribadi Anda dan letakkan di .envfile Anda, seperti yang dilakukan di atas.
9. Pergi ke Alchemy dan buat proyek baru di testnet pilihan (yaitu, kovan atau Rinkeby)
10. Ambil URL Anda yang terkait dengan testnet, dan letakkan di .envfile Anda.
11. Pastikan Anda memiliki testnet ETH di akun Anda. Anda bisa mendapatkan beberapa di sini . Anda harus mendapatkan testnet ETH untuk testnet yang sama dengan yang Anda buat proyek di Alkimia (yaitu, Rinkeby atau Kovan)
12. Run
node deploy.js

Lesson 6: Hardhat Simple Storage

Berikut adalah langkah yang perlu dilakukan untuk melakukan Hardhat Simle Storage.

Persyaratan

- git
Anda akan tahu bahwa Anda melakukannya dengan benar jika Anda dapat berlari git --version dan Anda melihat respons seperti version x.x.x
- Nodejs
Anda akan tahu bahwa Anda telah menginstal nodejs dengan benar jika Anda dapat menjalankan:
- node --version dan dapatkan output seperti: vx.x.x
- Benang bukannya npm
Anda akan tahu bahwa Anda telah memasang benang dengan benar jika Anda dapat menjalankan:
- yarn --version dan dapatkan output seperti: x.x.x
Anda mungkin perlu menginstalnya dengan npm atau corepack

```
git clone https://github.com/PatrickAlphaC/hardhat-simple-storage-fcc
cd hardhat-simple-storage-fcc
yarn
yarn hardhat
```

Used :

```
npx hardhat run scripts/deploy.js
```

Pengujian :

```
npx hardhat test
```

Test :

```
npx hardhat coverage
```

Perkiraan Gas :

```
npx hardhat test
```

Jika Anda ingin menjalankan jaringan hardhat lokal Anda sendiri, Anda dapat menjalankan:

```
npx hardhat node
```

Dan kemudian di terminal yang berbeda,

```
npx hardhat run scripts/deploy.js --network localhost
```

Dan Anda akan melihat transaksi terjadi di terminal Anda yang sedang berjalan

```
npx hardhat node
```

Jika Anda menerapkan ke testnet atau mainnet, Anda dapat memverifikasinya jika Anda mendapatkan Kunci API dari Etherscan dan menetapkannya sebagai variabel lingkungan bernama ETHERSCAN_API_KEY. Anda dapat memasukkannya ke dalam .envfile Anda seperti yang terlihat di file .env.example.

Dalam kondisi saat ini, jika Anda memiliki kunci api yang disetel, itu akan memverifikasi kontrak rinkeby secara otomatis!

Namun, Anda dapat memverifikasi secara manual dengan:

```
npx hardhat verify --constructor-args arguments.js DEPLOYED_CONTRACT_ADDRESS
```

Lesson 7: Hardhat Fund Me

Berikut adalah langkah yang perlu dilakukan untuk melakukan Hardhat Fund Me..

Persyaratan

- git
Anda akan tahu bahwa Anda melakukannya dengan benar jika Anda dapat berlari git --version dan Anda melihat respons seperti version x.x.x
- Nodejs
Anda akan tahu bahwa Anda telah menginstal nodejs dengan benar jika Anda dapat menjalankan:
- node --version dan dapatkan output seperti: vx.x.x
- Benang bukannya npm
Anda akan tahu bahwa Anda telah memasang benang dengan benar jika Anda dapat menjalankan:
- yarn --version dan dapatkan output seperti: x.x.x
Anda mungkin perlu menginstalnya dengan npm atau corepack

```
git clone https://github.com/PatrickAlphaC/hardhat-fund-me-fcc
cd hardhat-fund-me-fcc
yarn
```

Used :

```
npx hardhat run scripts/deploy.js
```

Pengujian :

```
npx hardhat test
```

Test :

```
npx hardhat coverage
```

Penerapan ke testnet atau mainnet

1. Atur variabel lingkungan
Anda ingin mengatur KOVAN_RPC_URL dan PRIVATE_KEY sebagai variabel lingkungan. Anda dapat menambahkannya ke .envfile, mirip dengan apa yang Anda lihat di .env.example.
 - ❖ PRIVATE_KEY: Kunci pribadi akun Anda (seperti dari metamask).
CATATAN: UNTUK PENGEMBANGAN, HARAP GUNAKAN KUNCI YANG TIDAK MEMILIKI DANA NYATA YANG TERKAIT DENGANNYA. Anda dapat mempelajari cara mengekspornya di sini.
 - ❖ KOVAN_RPC_URL: Ini adalah url dari simpul testnet kovan yang sedang Anda kerjakan. Anda bisa mendapatkan pengaturan dengan satu gratis dari Alkimia
2. Dapatkan testnet ETH
Buka faucets.chain.link dan dapatkan beberapa tesnet ETH. Anda akan melihat ETH muncul di metamask Anda.
3. Menyebarluaskan

```
yarn hardhat deploy --network kovan
```

Skrip

Setelah menyebarkan ke testnet atau jaringan lokal, Anda dapat menjalankan skrip.

```
yarn hardhat run scripts/fund.js atau yarn hardhat run scripts/withdraw.js
```

Perkiraan Gas :

```
npx hardhat test
```

Jika Anda ingin menjalankan jaringan hardhat lokal Anda sendiri, Anda dapat menjalankan:

```
npx hardhat node
```

Dan kemudian di terminal yang berbeda,

```
npx hardhat run scripts/deploy.js --network localhost
```

Dan Anda akan melihat transaksi terjadi di terminal Anda yang sedang berjalan

```
npx hardhat node
```

Untuk mendapatkan estimasi biaya gas dalam USD, Anda memerlukan COINMARKETCAP_API_KEY variabel lingkungan. Anda bisa mendapatkannya secara gratis dari CoinMarketCap

coinmarketcap: COINMARKETCAP_API_KEY, Kemudian, batalkan komentar pada baris hardhat.config.js untuk mendapatkan estimasi USD. Perhatikan saja, setiap kali Anda menjalankan pengujian, itu akan menggunakan panggilan API, jadi mungkin masuk akal untuk menonaktifkan penggunaan coinmarketcap sampai Anda membutuhkannya. Anda dapat menonaktifkannya dengan hanya mengomentari baris kembali.

Namun, Anda dapat memverifikasi secara manual dengan:

```
npx hardhat verify --constructor-args arguments.js DEPLOYED_CONTRACT_ADDRESS
```

Linting

Untuk memeriksa pemformatan linting/kode:

```
yarn lint
```

Atau untuk memperbaiki

```
yarn lint:fix
```

Memformat

```
yarn format
```

Lesson 8: HTML / Javascript Fund Me (Full Stack / Front End)

- Folder html-fund-me-fcc

Pertama, buatlah folder html-fund-me-fcc, jika saat ini berada di folder hardhat-fund-me-fcc keluar dari folder tersebut terlebih dahulu:

```
[./hardhat-fund-me-fcc]$ cd ..  
[./] $ mkdir html-fund-me-fcc  
[./] $ cd ./html-fund-me-fcc
```

- HTML setup

Kemudian buatlah file html sederhana dengan nama file index.html:

```
index.html
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Fund Me App</title>
</head>
<body>
  Hello!
</body>
</html>
```

Selanjutnya install server yang nantinya berfungsi untuk menjalankan dan melihat langsung file html di browser.

```
[./html-fund-me-fcc] $ yarn add --dev http-server
```

Setelah server terinstall maka selanjutnya jalankan server tersebut.

```
[./html-fund-me-fcc] $ yarn http-server
```

```
Available on:
  http://127.0.0.1:8080
  http://172.30.134.252:8080
Hit CTRL-C to stop the server
```

Kemudian pilih salah satu http untuk di running. disini kami menggunakan http://127.0.0.1:8080. Setelah server dijalankan dan memilih http maka index.html dapat diakses melalui browser.



Hello!

- Connecting HTML to Metamask

Untuk membuat connecting dengan metamask maka kita perlu membuat perintah java script.

```
index.html
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Fund Me App</title>
</head>
<body>
```

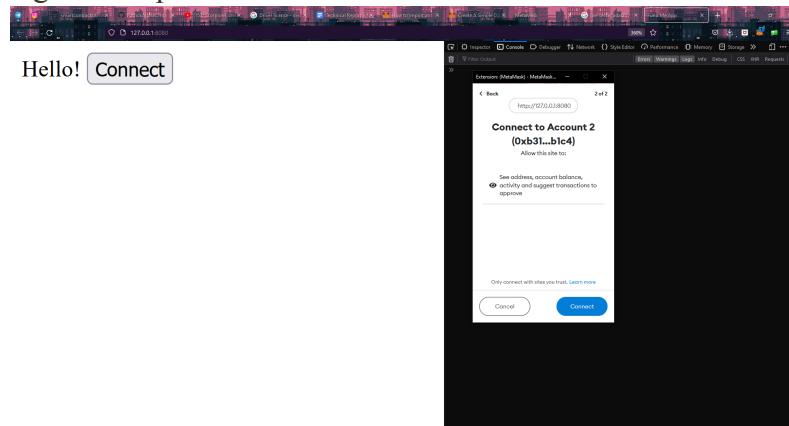
```

Hello!
<script>
async function connect() {
  if (typeof window.ethereum !== "undefined") {
    await window.ethereum.request({ method: "eth_requestAccounts" });
    document.getElementById("connectButton").innerHTML = "Connected!";
  } else {
    document.getElementById("connectButton").innerHTML =
      "Please install metamask!";
  }
}
</script>
<button id="connectButton" onclick="connect()">Connect</button>
</body>
</html>

```

- **window.ethereum** berfungsi untuk melakukan pengecekan apakah metamask telah terinstal di browser.
- **window.ethereum.request({ method: "eth_requestAccounts" })** berfungsi untuk mengecek autentikasi metamask

Setelah itu pergi ke browser klik tombol ‘connect’. maka nantinya browser akan menghubungkan ke aplikasi metamask.



- Javascript in its own files

Agar script terlihat rapi di file html maka perlu pisah script html dan javascript dengan cara membuat file javascript sendiri. Buatlah file index.js untuk memindahkan script javascript di html.

index.js

```

async function connect() {
  if (typeof window.ethereum !== "undefined") {
    await window.ethereum.request({ method: "eth_requestAccounts" });
    document.getElementById("connectButton").innerHTML = "Connected!";
  } else {
    document.getElementById("connectButton").innerHTML =

```

```
"Please install metamask!";  
}  
}
```

untuk menghubungkan file html dengan javascript maka perlu membuat kode seperti dibawah ini:

index.html

```
<!DOCTYPE html>  
//...  
<script src="./index.js" type="text/javascript"></script>  
<button id="connectButton" onclick="connect()">Connect</button>  
</body>  
</html>
```

- o ES6 (Front End JS) vs NodeJS

Buatlah file ethereum agar nantinya terbaca nilai java script. file etherium dapat di download [disini](#).

setelah file ethereum di download maka perlu menghubungkan file ethereum dengan javascript.

index.js

```
import { ethers } from "./ethers-5.6.esm.min.js"  
  
async function connect() {  
    if (typeof window.ethereum !== "undefined") {  
        await window.ethereum.request({ method: "eth_requestAccounts" })  
        connectButton.innerHTML = "Connected!"  
    }  
}
```

Kemudian buatlah fungsi funding (pendanaan) di java script.

index.js

```
//...  
async function fund(ethAmount) {  
    console.log(`Funding with ${ethAmount}...`)  
    if (typeof window.ethereum !== "undefined") {  
        // provider / connection to the blockchain  
        // signer / wallet / someone with some gas  
        // contract that we are interacting with  
        // ^ ABI & Address  
    }  
}
```

Setelah membuat fungsi funding, buatlah tombol untuk mengakses fungsi fund. namun Agar bisa terhubung dengan file ethereum maka kita perlu mengubah pemanggilan javascript.

```
index.html
```

```
//...
<script src="./index.js" type="module"></script>
<button id="connectButton">Connect</button>
<button id="fundButton">Fund</button>

</body>
//...
```

Karena pemanggilan berubah menjadi modul maka pemanggil tombol koneksi dan tombol funding berada di java script.

```
index.js
```

```
import { ethers } from "./ethers-5.6.esm.min.js"

const connectButton = document.getElementById("connectButton")
const fundButton = document.getElementById("fundButton")

connectButton.onclick = connect
fundButton.onclick = fund

async function connect() {
  if (typeof window.ethereum !== "undefined") {
    await window.ethereum.request({ method: "eth_requestAccounts" })
    connectButton.innerHTML = "Connected!"
  } else {
    connectButton.innerHTML = "Please install metamask!"
  }
}
//...
```

- Sending a Transaction From a Website

Selanjutnya buat file constant.js untuk membuat contract address dan ABI. file contract didapat dari **hardhat node**. Sedangkan data ABI berasal dari

hardhat-fund-me-fcc/artifacts/contracts/FundMe.sol/FundMe.JSON

Data ContractAddress:

```
[./hardhat-fund-me-fcc]$ yarn hardhat node
```

```

hamza@fatihulhaq@DESKTOP-7GU68RR:~/Blockchain/lesson8/hardhat-fund-me-fcc$ yarn hardhat node
yarn run v1.22.19
warning package.json: No license field
g /home/hamza@fatihulhaq/Blockchain/lesson8/hardhat-fund-me-fcc/node_modules/.bin/hardhat node
Nothing to compile
Local network detected! Deploying mocks...
deploying "Mock3Aggregator" (tx: 0x2f60bd4cba5dffe33cd22380f4891cfadb7f13aad763bb084e8a1c3336b892f9)...: deployed at 0x5fb082315678afeb367f032d93f642f64180aa3 with 569635 gas
Mocks Deployed!
You are deploying to a local network, you'll need a local network running to interact
Please run 'npx hardhat console' to interact with the deployed smart contracts!
-----
Deploying FundMe and waiting for confirmations...
deploying "FundMe" (tx: 0x0ef977a689cc4c4275285764be7255faad25c489428eafa32cc266562650125)...: deployed at 0xe7f1725e7734ce288f8367e1bb143e90bb3f0512 with 105863 gas
FundMe deployed at 0xe7f1725e7734ce288f8367e1bb143e90bb3f0512
-----
Deploying FundWithStorage and waiting for confirmations...
deploying "FundWithStorage" (tx: 0x716a3c50a86400e371732dfb4373b28696df0e6aa79a5b0046298cb2f7b18d8c)...: deployed at 0x9fe46736679d2d9a65f0992f2272dE9f3c7fa6e0 with 227342 gas

```

Data ABI :

```

① FundMe.json ×
hardhat-fund-me-fcc>artifacts>contracts>FundMe.sol>②
5   "abi": [
6     {
7       "inputs": [
8         {
9           "internalType": "address",
10          "name": "priceFeed",
11          "type": "address"
12        }
13      ],
14      "stateMutability": "nonpayable",
15      "type": "constructor"
16    },
17    {
18      "inputs": [],
19      "name": "FundMe__NotOwner",
20      "type": "error"
21    },
22    {
23      "inputs": [],
24      "name": "MINIMUM_USD",
25      "outputs": [
26        {
27          "internalType": "uint256",
28          "name": "",
29          "type": "uint256"
30        }
31      ],
32      "stateMutability": "view",
33      "type": "function"
34    }

```

constant.js

```

export const contractAddress = "0xe7f1725e7734ce288f8367e1bb143e90bb3f0512"
export const abi = [
  {
    inputs: [
      {
        internalType: "address",
        name: "priceFeed",
        type: "address",
      },
    ],
    stateMutability: "nonpayable",
    type: "constructor",
  },
]

```

```
{  
  inputs: [],  
  name: "FundMe__NotOwner",  
  type: "error",  
},  
//...
```

Selanjutnya hubungkan file **constant.js** dengan file **index.js**.

index.js

```
import { ethers } from "./ethers-5.6.esm.min.js"  
import { abi, contractAddress } from "./constants.js"  
  
const connectButton = document.getElementById("connectButton")  
const fundButton = document.getElementById("fundButton")  
  
//...
```

Selanjutnya buat provider, signer, contract dan respons transaksi di fungsi funding. dan untuk sementara funding etherium amountnya ditetapkan 77eth.

index.js

```
//...  
async function fund() {  
  const ethAmount = "77"  
  console.log(`Funding with ${ethAmount}...`)  
  if (typeof window.ethereum !== "undefined") {  
    // provider / connection to the blockchain  
    // signer / wallet / someone with some gas  
    // contract that we are interacting with  
    // ^ ABI & Address  
    const provider = new ethers.providers.Web3Provider(window.ethereum)  
    const signer = provider.getSigner()  
    const contract = new ethers.Contract(contractAddress, abi, signer)  
    try {  
      const transactionResponse = await contract.fund({  
        value: ethers.utils.parseEther(ethAmount),  
      })  
      } catch (error) {  
        console.log(error)  
      }  
    }  
}
```

kemudian tambahkan Network pada Metamask

The screenshot shows the Metamask Settings interface with the 'Networks' tab selected. A sub-menu item 'Add a network' is highlighted. The main area displays fields for adding a new network:

- Network Name:** Hardhat-localhost
- New RPC URL:** http://127.0.0.1:8545/
- Chain ID:** 31337
- Currency Symbol:** ETH

A note at the top right states: "A malicious network provider can lie about the state of the blockchain and record your network activity. Only add custom networks you trust."

Selanjutnya masukkan akun Hardhat ke Metamask:

1. Copy private key dari account yang telah dibuat dari hardhat node

```
hamzahfatihulhaq@DESKTOP-7GUG8RR:~/Blockchain/lesson8/hardhat-fund-me-fcc$ yarn hardhat node
yarn run v1.22.19
warning package.json: No license field

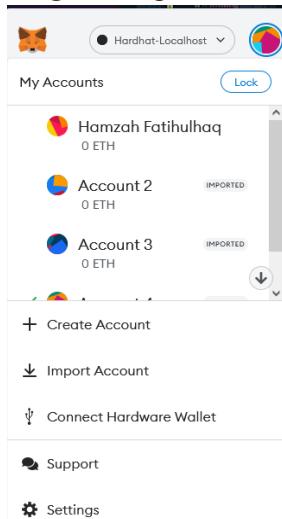
Accounts
=====
WARNING: These accounts, and their private keys, are publicly known.
Any funds sent to them on Mainnet or any other live network WILL BE LOST.

Account #0: 0xf39fd6e51aad88f6f4ce6ab8827279cfffb92266 (10000 ETH)
Private Key: 0xac974bec39a17e36ba46b4d238ff944bacb478cbef5efcae784d7bf4f2ff80

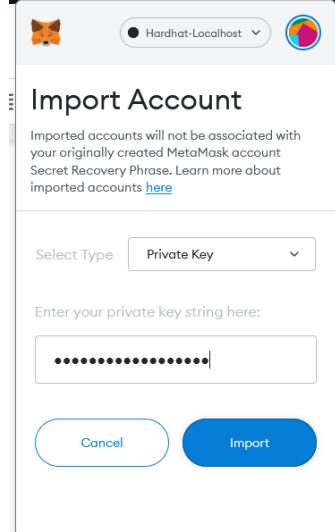
Account #1: 0x70997970c51812dc3a010c7d01b50e0d17dc79c8 (10000 ETH)
Private Key: 0x59c6995e998f97a5a0044966f0945389dc9e86dae88c7a8412f4603b6b78690d

Account #2: 0x3c44cdedb6a900fa2b585dd299e03d12fa4293bc (10000 ETH)
Private Key: 0x5de4111afaf1a4b94908f83103eb1f1706367c2e68ca870fc3fb9a804cdab365a
```

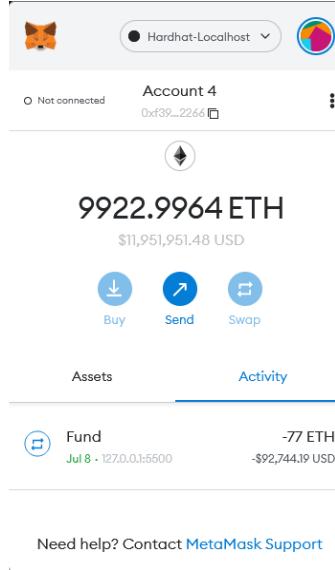
2. Pergi ke Import Account



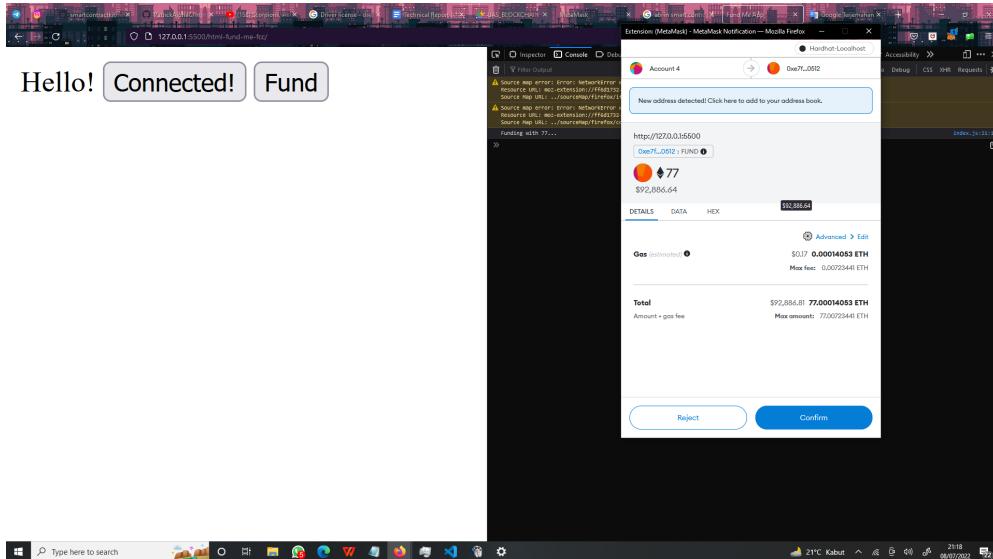
3. Paste private key kedalam dan lakukan import



4. Maka sekarang dapat menggunakan akun hardhat di metamask



Dan sekarang kita dapat melakukan funding di index.html



- Listening for Events and Completed Transactions
Buatlah fungsi transaksi mining di index.js.

```
index.js
```

```
//...
function listenForTransactionMine(transactionResponse, provider) {
  console.log(`Mining ${transactionResponse.hash}...`)
  return new Promise((resolve, reject) => {
    provider.once(transactionResponse.hash, (transactionReceipt) => {
      console.log(`Completed with ${transactionReceipt.confirmations}`)
      resolve()
    })
  })
}
```

Kemudian pada bagian funding panggil fungsi listenForTransactionMine.

```
index.js
```

```
//...
async function fund() {
  //...
  try {
    const transactionResponse = await contract.fund({
      value: ethers.utils.parseEther(ethAmount),
    })

    await listenForTransactionMine(transactionResponse, provider)
    console.log("Done!")
  } catch (error) {
    console.log(error)
  }
}
//...
```

Sekarang dapat melihat transaksi mining dengan menekan tombol funding

```
Funding with 77...
Mining 0x400f2e683119baa2169188e7ae1177802637cbafa2afe5c3a4275b2b1879e076...
Completed with 1 confirmations
Done!
```

- Input Form
Selanjutnya buat input form di index.html. Input form berfungsi nilai ethereum dapat diinputkan user.

index.html

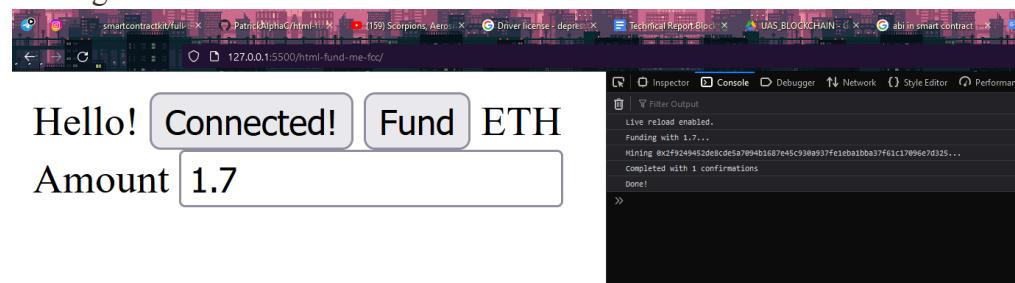
```
//...
<button id="fundButton">Fund</button>
<!-- form -->
<label for="fund">ETH Amount</label>
<input id="ethAmount" placeholder="0.1" />
</body>
//...
```

dan ubah kode ethAmount pada index.js untuk dapat terhubung dengan form input di html

index.js

```
//...
async function fund() {
  const ethAmount = document.getElementById("ethAmount").value
  console.log(`Funding with ${ethAmount}...`)
  if (typeof window.ethereum !== "undefined") {
    const provider = new
ethers.providers.Web3Provider(window.ethereum)
    const signer = provider.getSigner()
  }
//...
```

Sekarang user dapat melakukan perubahan nilai Etherium untuk nantinya di funding



- Reading form the blockchain
Buatlah fungsi balance di index.js

index.js

```
//...
async function connect() {
  //...
}
async function getBalance() {
  if (typeof window.ethereum !== "undefined") {
```

```

    const provider = new ethers.providers.Web3Provider(window.ethereum)
    const balance = await provider.getBalance(contractAddress)
    console.log(ethers.utils.formatEther(balance))
}

async function fund() {
//...

```

kemudian buat tombol balancing untuk memanggil fungsi getBalance.

index.html

```

//...
<button id="connectButton">Connect</button>
<button id="balanceButton">getBalance</button>
<!-- form -->
<label for="fund">ETH Amount</label>
//...

```

index.js

```

//...

const connectButton = document.getElementById("connectButton")
const fundButton = document.getElementById("fundButton")
const balanceButton = document.getElementById("balanceButton")

connectButton.onclick = connect
fundButton.onclick = fund
balanceButton.onclick = getBalance

//...

```

- Withdraw Function
Buatlah fungsi withdraw di index.js

index.js

```

//...
async function withdraw() {
  if (typeof window.ethereum !== "undefined") {
    console.log('Withdrawning...')
    const provider = new ethers.providers.Web3Provider(window.ethereum)

```

```

const signer = provider.getSigner()
const contract = new ethers.Contract(contractAddress, abi, signer)
try {
    const transactionResponse = await contract.withdraw()
    await listenForTransactionMine(transactionResponse, provider)
} catch (error) {
    console.log(error)
}
}
}

```

Buatlah tombol withdraw untuk memanggil fungsi withdraw

index.html

```

//...
<button id="balanceButton">getBalance</button>
<button id="withdrawButton">withdraw</button>
<!-- form -->
<label for="fund">ETH Amount</label>
//...

```

index.js

```

//...
const balanceButton = document.getElementById("balanceButton")
const withdrawButton = document.getElementById("withdrawButton")

//...
balanceButton.onclick = getBalance
withdrawButton.onclick = withdraw

```

Lesson 9: Hardhat Smart Contract Lottery

- Hardhat Setup

Buatlah setup hardhat:

```

$ mkdir hardhat-smartcontract-lottery-fcc
$ cd hardhat-smartcontract-lottery-fcc
[ hardhat-smartcontract-lottery-fcc] $ yarn add --dev hardhat
[ hardhat-smartcontract-lottery-fcc] $ yarn hardhat
>> create an empty hardhat config js
[      hardhat-smartcontract-lottery-fcc]      $      yarn      add      --dev
[@nomiclabs/hardhat-ethers@npm:hardhat-deploy-ethers

```

```
@nomiclabs/hardhat-etherscan @nomiclabs/hardhat-waffle chai ethereum-waffle hardhat hardhat-contract-sizer hardhat-deploy hardhat-gas-reporter prettier prettier-plugin-solidity solhint solidity-coverage dotenv
```

- Typescript

Jika ingin mendapatkan TypeScript dan Anda mengkloning versi javascript, jalankan saja:

```
git checkout typescript
```

- Penggunaan

```
yarn hardhat deploy
```

- Pengujian

```
yarn hardhat test
```

- Cakupan Tes

```
yarn hardhat coverage
```

Penerapan ke testnet atau mainnet

1. Atur variabel lingkungan

Jika ingin mengatur RINKEBY_RPC_URL dan PRIVATE_KEY sebagai variabel lingkungan. Anda dapat menambahkannya ke .envfile, mirip dengan apa yang Anda lihat di .env.example.

- PRIVATE_KEY: Kunci pribadi akun Anda (seperti dari metamask).
CATATAN: UNTUK PENGEMBANGAN, HARAP GUNAKAN KUNCI YANG TIDAK MEMILIKI DANA NYATA YANG TERKAIT DENGANNYA.
 - Anda dapat mempelajari cara mengeksporinya di sini .
- RINKEBY_RPC_URL: Ini adalah url dari node rinkeby testnet yang sedang Anda kerjakan. Anda bisa mendapatkan pengaturan dengan satu gratis dari Alkimia

2. Dapatkan testnet ETH

Buka faucets.chain.link dan dapatkan beberapa tesnet ETH & LINK. Anda akan melihat ETH dan LINK muncul di metamask Anda. Anda dapat membaca lebih lanjut tentang menyiapkan dompet Anda dengan LINK.

3. Siapkan ID Berlangganan Chainlink VRF

Buka vrf.chain.link dan siapkan langganan baru, dan dapatkan subscriptionId. Anda dapat menggunakan kembali langganan lama jika sudah memilikinya. Anda dapat mengikuti petunjuk jika tersesat. Anda harus meninggalkan langkah ini dengan:

1. ID berlangganan
2. Langganan Anda harus didanai dengan LINK
3. Menyebarluaskan

Di Anda helper-hardhat-config.js tambahkan Anda subscriptionId di bawah bagian chainId yang Anda gunakan (alias, jika Anda menggunakan rinkeby, tambahkan Anda subscriptionId di subscriptionId bidang di bawah 4 bagian tersebut.)

Lalu run:

```
yarn hardhat deploy --network rinkeby
```

Dan salin / ingat alamat kontrak.

4. Tambahkan alamat kontrak Anda sebagai Konsumen Chainlink VRF
Kembali ke [vrf.chain.link](#) dan di bawah langganan Anda tambahkan Add consumer dan tambahkan alamat kontrak Anda. Anda juga harus mendanai kontrak dengan minimal 1 LINK.
5. Daftarkan Pemeliharaan Penjaga Chainlink
[Anda dapat mengikuti dokumentasi jika Anda tersesat.](#)

Buka [keepers.chain.link](#) dan daftarkan pemeliharaan baru. UI Anda akan terlihat seperti ini setelah selesai:

6. Masukkan undian Anda!

Kontrak Anda sekarang siap untuk menjadi lotere acak yang dapat diverifikasi secara otonom dan tidak dapat dirusak. Masukkan lotere dengan menjalankan:

```
yarn hardhat run scripts/enter.js --network rinkeby
```

- Verifikasi di etherscan

Jika Anda menerapkan ke testnet atau mainnet, Anda dapat memverifikasinya jika Anda mendapatkan [Kunci API](#) dari Etherscan dan menetapkannya sebagai variabel lingkungan bernama ETHERSCAN_API_KEY. Anda dapat memasukkannya ke dalam .envfile Anda seperti yang terlihat di file .env.example.

Dalam keadaan saat ini, jika Anda memiliki set kunci api, itu akan memverifikasi kontrak kovan secara otomatis!

Namun, Anda dapat memverifikasi secara manual dengan:

```
yarn hardhat verify --constructor-args arguments.js  
DEPLOYED_CONTRACT_ADDRESS
```

Lesson 10: NextJS Smart Contract Lottery (Full Stack / Front End)

Decentralized Lottery

9999.99975790 0x7099...dc79c8 🎉

Lottery

Enter Raffle

Entrance Fee: 0.1 ETH

The current number of players is: 1

The most previous winner was: 0x70997970C51812dc3A010C7d01b50e0d17dc79C8

```
git clone https://github.com/PatrickAlphaC/nextjs-smartcontract-lottery-fcc
cd nextjs-smartcontract-lottery-fcc
yarn
yarn dev
```

- Typescript

Jika ingin mendapatkan TypeScript dan mengkloning versi javascript, jalankan saja:

```
git checkout typescript
```

- Formatting in VSCode

Untuk memiliki ekstensi VSCode format otomatis .jsx dan .tsx yang lebih cantik, tambahkan yang berikut ini ke file settings.json:

```
"[typescriptreact)": {
  "editor.defaultFormatter": "esbenp.prettier-vscode"
},
"[javascriptreact)": {
  "editor.defaultFormatter": "esbenp.prettier-vscode"
}
```

- Useage

1. Jalankan blockchain lokal dengan kode lotere

```
git clone https://github.com/PatrickAlphaC/hardhat-fund-me-fcc
cd hardhat-fund-me-fcc
yarn
yarn hardhat node
```

2. Tambahkan jaringan hardhat ke metamask/dompet Anda

Dapatkan RPC_URL node hh Anda (biasanya <http://127.0.0.1:8545/>). Buka dompet dan tambahkan jaringan baru. Lihat petunjuk di sini.

Nama Jaringan: Hardhat-Localhost

URL RPC baru: http://127.0.0.1:8545/

ID Rantai: 31337

Simbol Mata Uang: ETH (atau GO)

Blokir URL Penjelajah: Tidak Ada

Idealnya, kemudian akan mengimpor salah satu akun dari hardhat ke dompet/metamask.

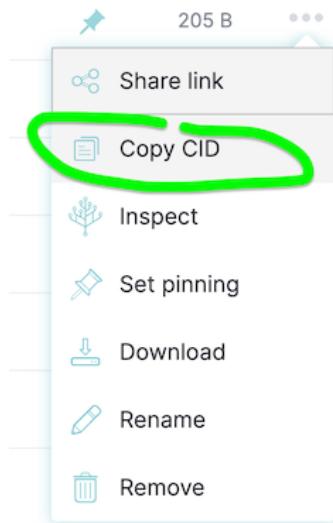
3. Run this code

Kembali di terminal yang berbeda dengan kode dari repo ini, jalankan:

```
yarn dev
```

4. Buka UI dan bersenang-senanglah!

Pergilah ke localhost Anda dan mainkan lotere!



o Deploying to IPFS

Buat kode statis.

```
yarn build
```

Eksport situs

```
yarn next export
```

Terapkan ke IPFS

- o Unduh desktop IPFS
- o Buka aplikasi desktop IPFS Anda
- o Pilih import dan pilih folder yang langkah di atas baru saja dibuat (harus keluar)

Salin CID folder yang Anda sematkan. Dapatkan pendamping IPFS untuk browser Anda (atau gunakan Brave Browser). Buka ipfs://YOUR_CID_HERE dan lihat situs ipfs Anda yang digunakan!

- o Linting

```
yarn lint
```

Lesson 11: Hardhat Starter Kit

Mengkloning dan menginstal dependensi. Setelah menginstal semua persyaratan, jalankan yang berikut ini:

```
git clone https://github.com/smartcontractkit/hardhat-starter-kit/  
cd hardhat-starter-kit
```

Kemudian:

```
yarn
```

Sekarang dapat melakukan staff!

```
yarn hardhat test
```

- Typescript

Untuk menggunakan TypeScript, jalankan:

```
git checkout typescript  
yarn
```

- Usage

Jika Anda menjalankan **yarn hardhat --help** Anda akan mendapatkan output dari semua tugas yang dapat Anda jalankan.

- Deploying Contracts

```
yarn hardhat deploy
```

Ini akan menyebarkan kontrak Anda ke jaringan lokal. Selain itu, jika di jaringan lokal, itu akan menyebarkan kontrak Chainlink tiruan untuk Anda berinteraksi. Jika Anda ingin berinteraksi dengan kontrak yang diterapkan, lewati ke Berinteraksi dengan Kontrak yang Diterapkan.

- Run a Local Network

Salah satu cara terbaik untuk menguji dan berinteraksi dengan kontrak pintar adalah dengan jaringan lokal. Untuk menjalankan jaringan lokal dengan semua kontrak Anda di dalamnya, jalankan yang berikut ini:

```
yarn hardhat node
```

Anda akan mendapatkan blockchain lokal, kunci pribadi, kontrak yang disebarluaskan (dari skrip folder penyebaran), dan titik akhir yang berpotensi ditambahkan ke dompet EVM.

- Using a Testnet or Live Network (like Mainnet or Polygon)

Di hardhat.config.js Anda, Anda akan melihat bagian seperti:

```
module.exports = {
  defaultNetwork: "hardhat",
  networks: {
```

Bagian file ini adalah tempat Anda menentukan jaringan mana yang ingin Anda gunakan untuk berinteraksi. Anda dapat membaca lebih lanjut tentang seluruh file di [dokumentasi hardhat](#).

Untuk berinteraksi dengan jaringan langsung atau uji coba, Anda memerlukan:

1. URL rpc
2. Kunci Pribadi
3. Token ETH & LINK (baik testnet atau nyata)

Mari kita lihat contoh pengaturan ini menggunakan testnet Rinkeby.

1. Rinkeby Ethereum Testnet Setup

Pertama, kita perlu mengatur variabel lingkungan. Kita dapat melakukannya dengan mengurnya di .envfile kita (buat jika tidak ada). Anda juga dapat membaca lebih lanjut tentang [variabel lingkungan](#) dari blog twilio yang ditautkan. Anda akan menemukan contoh tampilan file ini di .env.example

PENTING: PASTIKAN ANDA TIDAK MENGUNGKAPKAN KUNCI YANG ANDA MASUKKAN DI .envFILE INI. Maksud saya, jangan mendorong mereka ke repo publik, dan tolong coba simpan kunci yang Anda gunakan dalam pengembangan yang tidak terkait dengan dana nyata apa pun.

2. Tetapkan RINKEBY_RPC_URL [variabel lingkungan Anda](#).

Anda bisa mendapatkannya secara gratis dari [Alchmey](#), [Infura](#), atau [Moralis](#). Ini adalah koneksi Anda ke blockchain.

3. Tetapkan PRIVATE_KEY variabel lingkungan Anda.

Ini adalah kunci pribadi Anda dari dompet Anda, yaitu [MetaMask](#). Ini diperlukan untuk menyebarkan kontrak ke jaringan publik. Anda dapat secara opsional mengatur MNEMONIC variabel lingkungan Anda dengan beberapa perubahan pada file hardhat.config.js.

Saat mengembangkan, praktik terbaik adalah menggunakan Metamask yang tidak terkait dengan uang sungguhan. Cara yang baik untuk melakukannya adalah dengan membuat profil browser baru (di Chrome, Brave, Firefox, dll) dan menginstal Metamask di browser itu, dan jangan pernah mengirim uang dompet ini.

Jangan komit dan dorong perubahan apa pun ke file .env yang mungkin berisi informasi sensitif, seperti kunci pribadi! Jika informasi ini mencapai repositori GitHub publik, seseorang dapat menggunakannya untuk memeriksa apakah Anda memiliki dana Mainnet di alamat dompet itu, dan mencurinya!

.env contoh:

```
RINKEBY_RPC_URL='www.infura.io/asdfadsfafdadf
PRIVATE_KEY='abcdef'
```

bash contoh

```
export RINKEBY_RPC_URL='www.infura.io/asdfadsfafdadf
export PRIVATE_KEY='abcdef'
```

Untuk jaringan lain seperti mainnet dan poligon, Anda dapat menggunakan variabel lingkungan yang berbeda untuk URL RPC dan kunci pribadi Anda. Lihat hardhat.config.js untuk mempelajari lebih lanjut.

1. Dapatkan beberapa Rinkeby Testnet ETH dan LINK
Pergilah ke [faucet Chainlink](#) dan dapatkan beberapa ETH dan LINK. Silakan ikuti [dokumentasi chainlink](#) jika tidak familiar.
2. Buat langganan VRF V2
[Buka Halaman Berlangganan VRF](#) dan buat langganan baru. Simpan ID langganan Anda dan masukkan ke dalam .envfile sebagai VRF_SUBSCRIPTION_ID
3. Menjalankan perintah
Anda sekarang harus siap! Anda dapat menjalankan perintah apa pun dan cukup lewati --network rinkeby sekarang!

Untuk menerapkan kontrak:

```
yarn hardhat deploy --network rinkeby
```

Untuk menjalankan pementasan tes testnet

- Forking

Jika Anda ingin menjalankan pengujian atau pada jaringan yang merupakan [jaringan bercabang](#)

- Tetapkan MAINNET_RPC_URL variabel lingkungan yang terhubung ke mainnet.
- Pilih nomor blokir untuk memilih status jaringan yang Anda forking dan atur sebagai FORKING_BLOCK_NUMBER variabel lingkungan. Jika diabaikan, itu akan menggunakan blok terbaru setiap kali yang dapat menyebabkan inkonsistensi pengujian.
- Setel enabled tanda ke true/ false untuk mengaktifkan/menonaktifkan fitur forking

```
forking: {  
    url: MAINNET_RPC_URL,  
    blockNumber: FORKING_BLOCK_NUMBER,  
    enabled: false,  
}
```

- Test

Pengujian terletak di direktori [pengujian](#), dan dibagi antara pengujian unit dan pengujian staging/testnet. Tes unit hanya boleh dijalankan di lingkungan lokal, dan tes pementasan hanya boleh dijalankan di lingkungan langsung.

Untuk menjalankan pengujian unit:

```
yarn test
```

Untuk menjalankan tes pementasan pada jaringan Rinkeby:

```
yarn test-staging
```

- Performance optimizations

Karena semua tes ditulis dengan cara yang independen satu sama lain, Anda dapat menghemat waktu dengan menjalankannya secara paralel. Pastikan bahwa AUTO_FUND=false di dalam .envfile. Ada beberapa batasan dengan pengujian paralel, baca selengkapnya [di sini](#)

Untuk menjalankan tes secara paralel:

```
yarn test --parallel
```

- Interacting with Deployed Contracts

Setelah menerapkan kontrak Anda, output penerapan akan memberi Anda alamat kontrak saat diterapkan. Anda kemudian dapat menggunakan alamat kontrak ini bersama dengan tugas Hardhat untuk melakukan operasi pada setiap kontrak.

- Chainlink Price Feeds

Kontrak konsumen Feed Harga memiliki satu tugas, untuk membaca harga terbaru dari kontrak feed harga tertentu

```
yarn hardhat read-price-feed --contract insert-contract-address-here --network network
```

- Request & Receive Data

Kontrak APIConsumer memiliki dua tugas, satu untuk meminta data eksternal berdasarkan serangkaian parameter, dan satu lagi untuk memeriksa untuk melihat apa hasil dari permintaan data tersebut. Kontrak ini perlu didanai dengan tautan terlebih dahulu:

```
yarn hardhat fund-link --contract insert-contract-address-here --network network
```

Setelah didanai, Anda dapat meminta data eksternal dengan meneruskan sejumlah parameter ke tugas permintaan-data. Parameter kontrak adalah wajib, sisanya opsional

```
yarn hardhat request-data --contract insert-contract-address-here --network network
```

Setelah Anda berhasil membuat permintaan untuk data eksternal, Anda dapat melihat hasilnya melalui tugas baca-data

```
yarn hardhat read-data --contract insert-contract-address-here --network network
```

- VRF Get a random number

Kontrak VRFCConsumer memiliki dua tugas, satu untuk meminta nomor acak, dan satu lagi untuk membaca hasil dari permintaan nomor acak. Untuk memulai, buka [Halaman Berlangganan VRF](#) dan buat langganan baru. Simpan ID langganan Anda dan masukkan ke dalam .envfile sebagai VRF SUBSCRIPTION ID:

```
VRF_SUBSCRIPTION_ID=subscription_id
```

Kemudian, gunakan konsumen kontrak VRF V2 Anda ke jaringan langganan terbaru Anda menggunakan id langganan sebagai argumen konstruktor.

```
yarn hardhat deploy --network network
```

Terakhir, Anda perlu membuka halaman langganan Anda sekali lagi dan menambahkan alamat kontrak yang diterapkan sebagai konsumen baru. Setelah selesai, Anda dapat melakukan permintaan VRF dengan tugas nomor-permintaan-acak:

```
yarn hardhat request-random-number --contract insert-contract-address-here --network network
```

Setelah Anda berhasil membuat permintaan untuk nomor acak, Anda dapat melihat hasilnya melalui tugas read-random-number:

```
yarn hardhat read-random-number --contract insert-contract-address-here  
--network network
```

- Keepers

Kontrak KeepersCounter adalah kontrak berkemampuan Chainlink Keepers sederhana yang hanya mempertahankan variabel penghitung yang bertambah setiap kali tugas performUpkeep dilakukan oleh Penjaga Chainlink. Setelah kontrak diterapkan, Anda harus menuju ke <https://keepers.chain.link/> untuk mendaftarkannya untuk pemeliharaan, kemudian Anda dapat menggunakan tugas di bawah ini untuk melihat variabel penghitung yang ditingkatkan oleh Penjaga Chainlink.

```
yarn hardhat read-keepers-counter --contract insert-contract-address-here  
--network network
```

- Verify on Etherscan

Anda akan membutuhkan ETHERSCAN_API_KEY variabel lingkungan. Anda bisa mendapatkannya dari situs [Etherscan API](#). Jika Anda telah menyetelnya, skrip penerapan Anda akan mencoba memverifikasinya secara default, tetapi jika Anda ingin memverifikasi secara manual, Anda dapat menjalankan:

```
yarn hardhat verify --network <NETWORK> <CONTRACT_ADDRESS>  
<CONSTRUCTOR_PARAMETERS>
```

contoh:

```
yarn hardhat verify --network rinkeby  
0x9279791897f112a41FfDa267ff7DbBC46b96c296  
"0x9326BFA02ADD2366b30bacB125260Af641031331"
```

- View Contracts Size

```
yarn run hardhat size-contracts
```

- Linting

Ini akan merusak kontrak pintar Anda.

```
yarn lint:fix
```

- Code Formatting

Ini akan memformat javascript dan soliditas Anda agar terlihat lebih bagus.

```
yarn format
```

- Pemformatan Kode

Ini akan memformat javascript dan soliditas Anda agar terlihat lebih bagus.

```
yarn format
```

- Memperkirakan Gas

Untuk memperkirakan gas, cukup setel REPORT_GASvariabel lingkungan ke true, lalu jalankan:

```
yarn hardhat test
```

Jika Anda ingin melihat harga gas dalam USD atau mata uang lainnya, tambahkan COINMARKETCAP_API_KEY dari [Coinmarketcap](#).

- Cakupan kode

Untuk melihat ukuran dalam persen sejauh mana kode sumber kontrak pintar dijalankan saat rangkaian pengujian tertentu dijalankan, ketik

```
yarn coverage
```

Kita akan menggunakan Echidna sebagai alat pengujian Fuzz. Anda harus menginstal [Docker](#) dengan setidaknya 8 GB memori virtual yang dialokasikan (Untuk memperbarui parameter ini, buka *Settings->Resources->Advanced->Memory*).

Untuk memulai jalankan instance Echidna

```
yarn fuzzing
```

Jika Anda menggunakannya untuk pertama kali, Anda harus menunggu Docker mengunduh image [eth-security-toolbox](#) untuk kami.

Untuk memulai lari Fuzzing

```
echidna-test      /src/contracts/test/fuzzing/KeepersCounterEchidnaTest.sol      --contract
KeepersCounterEchidnaTest --config /src/contracts/test/fuzzing/config.yaml
```

Lesson 12: Hardhat ERC20s

```
git clone https://github.com/PatrickAlphaC/hardhat-erc20-fcc
cd hardhat-erc20-fcc
yarn
```

- Penggunaan

Menyebarluaskan:

```
yarn hardhat deploy
```

- Penerapan ke testnet atau mainnet
- 1. Atur variabel lingkungan

Anda ingin mengatur KOVAN_RPC_URL dan PRIVATE_KEY sebagai variabel lingkungan. Anda dapat menambahkannya ke .envfile, mirip dengan apa yang Anda lihat di .env.example.

- PRIVATE_KEY: Kunci pribadi akun Anda (seperti dari [metamask](#)).
CATATAN: UNTUK PENGEMBANGAN, HARAP GUNAKAN KUNCI YANG TIDAK MEMILIKI DANA NYATA YANG TERKAIT DENGANNYA.

- Anda dapat [mempelajari cara mengekspornya di sini](#).
 - KOVAN_RPC_URL: Ini adalah url dari simpul testnet kovan yang sedang Anda kerjakan. Anda bisa mendapatkan pengaturan dengan satu gratis dari [Alkimia](#)
2. Dapatkan testnet ETH
Buka faucets.chain.link dan dapatkan beberapa tesnet ETH. Anda akan melihat ETH muncul di metamask Anda.
 3. Menyebarkan

```
yarn hardhat deploy --network kovan
```

- Verifikasi di etherscan

Jika Anda menerapkan ke testnet atau mainnet, Anda dapat memverifikasinya jika Anda mendapatkan [Kunci API](#) dari Etherscan dan menetapkannya sebagai variabel lingkungan bernama ETHERSCAN_API_KEY. Anda dapat memasukkannya ke dalam .envfile Anda seperti yang terlihat di file .env.example.

Dalam keadaan saat ini, jika Anda memiliki set kunci api, itu akan memverifikasi kontrak kovan secara otomatis!

Namun, Anda dapat memverifikasi secara manual dengan:

```
yarn hardhat verify --constructor-args arguments
DEPLOYED_CONTRACT_ADDRESS
```

Lesson 13: Hardhat DeFi & Aave

```
git clone https://github.com/PatrickAlphaC/hardhat-defi-fcc
cd hardhat-defi-fcc
yarn
```

TypeScript

```
git checkout typescript
```

- Penggunaan

Repo ini membutuhkan penyedia rpc mainnet, tapi jangan khawatir! Anda tidak perlu mengeluarkan uang sungguhan. Kita akan menjadi forkingmainnet, dan berpura-pura seolah-olah kita sedang berinteraksi dengan kontrak mainnet.

Yang Anda perlukan hanyalah mengatur MAINNET_RPC_URL variabel lingkungan dalam .envfile yang Anda buat. Anda bisa mendapatkan pengaturan dengan satu gratis dari [Alkimia](#)

run:

```
yarn hardhat run scripts/aaveBorrow.js
```

- Berjalan di testnet atau mainnet

1. Atur variabel lingkungan

Anda ingin mengatur KOVAN_RPC_URL dan PRIVATE_KEY sebagai variabel

lingkungan. Anda dapat menambahkannya ke .envfile, mirip dengan apa yang Anda lihat di .env.example.

- PRIVATE_KEY: Kunci pribadi akun Anda (seperti dari [metamask](#)).
CATATAN: UNTUK PENGEMBANGAN, HARAP GUNAKAN KUNCI YANG TIDAK MEMILIKI DANA NYATA YANG TERKAIT DENGANNYA.
 - Anda dapat [mempelajari cara mengekspornya di sini](#) .
- KOVAN_RPC_URL: Ini adalah url dari simpul testnet kovan yang sedang Anda kerjakan. Anda bisa mendapatkan pengaturan dengan satu gratis dari [Alkimia](#)

2. Dapatkan testnet ETH

[Buka faucets.chain.link](#) dan dapatkan beberapa tesnet ETH. Anda akan melihat ETH muncul di metamask Anda.

3. Run

```
yarn hardhat run scripts/aaveBorrow.js --network kovan
```

Lesson 14: Hardhat NFTs



```
git clone https://github.com/PatrickAlphaC/hardhat-nft-fcc  
cd hardhat-nft-fcc  
yarn
```

○ Typescript

Jika ingin mendapatkan TypeScript dan Anda mengkloning versi javascript, jalankan saja:

```
git checkout typescript
```

○ Penggunaan

```
yarn hardhat deploy
```

○ Pengujian

```
yarn hardhat test
```

○ Cakupan Tes

```
yarn hardhat coverage
```

- Penerapan ke testnet atau mainnet
- 1. Atur variabel lingkungan

Anda ingin mengatur RINKEBY_RPC_URL dan PRIVATE_KEY sebagai variabel lingkungan. Anda dapat menambahkannya ke .envfile, mirip dengan apa yang Anda lihat di .env.example.

- PRIVATE_KEY: Kunci pribadi akun Anda (seperti dari [metamask](#)).
CATATAN: UNTUK PENGEMBANGAN, HARAP GUNAKAN KUNCI YANG TIDAK MEMILIKI DANA NYATA YANG TERKAIT DENGANNYA.
 - Anda dapat [mempelajari cara mengekspornya di sini](#).
- RINKEBY_RPC_URL: Ini adalah url dari node rinkeby testnet yang sedang Anda kerjakan. Anda bisa mendapatkan pengaturan dengan satu gratis dari [Alkimia](#)

2. Dapatkan testnet ETH

Buka [faucets.chain.link](#) dan dapatkan beberapa tesnet ETH & LINK. Anda akan melihat ETH dan LINK muncul di metamask Anda. [Anda dapat membaca lebih lanjut tentang menyiapkan dompet Anda dengan LINK](#).

3. Siapkan ID Berlangganan Chainlink VRF

Buka [vrf.chain.link](#) dan siapkan langganan baru, dan dapatkan subscriptionId. Anda dapat menggunakan kembali langganan lama jika sudah memilikinya.

[Anda dapat mengikuti petunjuk](#) jika tersesat. Anda harus meninggalkan langkah ini dengan:

1. ID berlangganan
2. Langganan Anda harus didanai dengan LINK
3. Menyebarluaskan

Di `helper-hardhat-config.tst` tambahkan Anda `subscriptionId` bawah bagian `chainId` yang Anda gunakan (alias, jika Anda menggunakan rinkeby, tambahkan Anda `subscriptionId` `subscriptionId` dibawah `4` bagian tersebut.)

Lalu run:

```
yarn hardhat deploy --network rinkeby --tags main
```

Kami hanya menyebarkan maintag, karena kami perlu menambahkan RandomIpfsNftkontrak kami sebagai konsumen.

4. Tambahkan alamat kontrak Anda sebagai Konsumen Chainlink VRF

Kembali ke [vrf.chain.link](#) dan di bawah langganan Anda tambahkan Add consumer dan tambahkan alamat kontrak Anda. Anda juga harus mendanai kontrak dengan minimal 1 LINK.

5. NFT mint

Lalu run:

```
yarn hardhat deploy --network rinkeby --tags mint
```

- Verifikasi di etherscan

Jika Anda menerapkan ke testnet atau mainnet, Anda dapat memverifikasinya jika Anda mendapatkan Kunci API dari Etherscan dan menetapkannya sebagai variabel lingkungan bernama ETHERSCAN_API_KEY. Anda dapat memasukkannya ke dalam .envfile Anda seperti yang terlihat di file .env.example.

Dalam keadaan saat ini, jika Anda memiliki set kunci api, itu akan memverifikasi kontrak kovan secara otomatis!

Namun, Anda dapat memverifikasi secara manual dengan:

```
yarn hardhat verify --constructor-args arguments.ts  
DEPLOYED_CONTRACT_ADDRESS
```

- Linting

Untuk memeriksa pemformatan linting/kode:

```
yarn lint
```

atau, untuk memperbaiki:

```
yarn lint:fix
```

Lesson 15: NextJS NFT Marketplace (Full Stack / Front End)

```
git clone https://github.com/PatrickAlphaC/hardhat-nft-marketplace-fcc  
cd hardhat-nextjs-nft-marketplace-fcc  
yarn
```

Setelah menginstal dependensi, mulai simpul di terminalnya sendiri dengan:

```
yarn hardhat node
```

- Connect your codebase to your moralis server

Siapkan moralis acara Anda . Anda memerlukan server moralis baru untuk memulai. Mendaftar untuk mendapatkan akun gratis di sini . Setelah pengaturan, perbarui / buat .envfile Anda. Anda dapat menggunakan .env.example sebagai boilerplate.

```
NEXT_PUBLIC_APP_ID=XXXX  
NEXT_PUBLIC_SERVER_URL=XXXX  
moralisApiKey=XXX  
moralisSubdomain=XXX  
masterKey=XXX  
chainId=31337
```

Dengan nilai dari akun Anda. Kemudian, dalam ./package.json pembaruan Anda, baris berikut:

```
"moralis:sync": "moralis-admin-cli connect-local-devchain --chain hardhat  
--moralisSubdomain XXX.usemoralis.com --frpcPath ./frp/frpc",  
"moralis:cloud": "moralis-admin-cli watch-cloud-folder --moralisSubdomain  
XXX.usemoralis.com --autoSave 1 --moralisCloudfolder ./cloudFunctions",  
"moralis:logs": "moralis-admin-cli get-logs --moralisSubdomain  
XXX.usemoralis.com"
```

"moralis:logs": "moralis-admin-cli get-logs --moralisSubdomain XXX.usemoralis.com"
Ganti XXX.usemoralis.com dengan subdomain Anda, suka 4444acatycat.usemoralis.com dan
perbarui jalur moralis:syncskrip ke instance Anda frp(diunduh sebagai bagian dari instruksi
Moralis "Devchain Proxy Server" yang disebutkan di atas)

- Instal secara global moralis-admin-cli

```
yarn global add moralis-admin-cli
```

- Siapkan proxy terbalik Moralis Anda

- Unduh kode proxy terbalik terbaru dari [FRP](#) dan tambahkan biner ke ./frp/frpc.
- Ganti konten Anda di frpc.ini, berdasarkan devchain Anda. Anda dapat menemukan
informasi di DevChain Proxy Server tab server moralis Anda.

Di beberapa Versi Windows, FRP dapat diblokir oleh firewall, gunakan saja rilis yang lebih
lama, misalnya frp_0.34.3_windows_386

Pemecahan Masalah Mac / Windows: <https://docs.moralis.io/faq#frpc>

Setelah Anda mendapatkan semua ini, Anda dapat menjalankan:

```
yarn moralis:sync
```

- Add your event listeners

```
node watchEvents.js
```

- Mint and List your NFT

Kembali ke direktori utama, jalankan:

```
yarn hardhat run scripts/mint-and-list-item.js --network localhost
```

- MStart your front end

Pada titik ini, Anda akan menjalankan beberapa terminal:

- Node Hardhat Anda
- Hardhat Node Anda disinkronkan dengan server moralis Anda
- Sinkronisasi Cloud Functions Anda
- Pencatatan Moral Anda

Dan Anda akan memiliki satu lagi untuk ujung depan Anda.

```
yarn run dev
```

- Minimal Quickstart

1. Kloning repo backend

```
git clone https://github.com/PatrickAlphaC/hardhat-nft-marketplace-fcc
cd hardhat-nextjs-nft-marketplace-fcc
yarn
yarn hardhat node
```

Biarkan terminal itu berjalan ^

2. Mengkloning frontend

```
git clone https://github.com/PatrickAlphaC/nextjs-nft-marketplace-moralis-fcc
cd nextjs-nft-marketplace-moralis-fcc
yarn
```

Siapkan Anda .env dengan info moralis dan perbarui Anda package.json dengan subdomain moralis.

3. Sinkronkan simpul hardhat Anda dengan moralis
Perbarui frpc.inifile Anda dengan apa yang Anda lihat di UI moralis.
Biarkan terminal ini berjalan:

```
yarn moralis:sync
```

4. Tonton acara && Perbarui fungsi cloud
Jalankan sekali:

```
yarn moralis:cloud
node watchEvents.js
```

5. Memancarkan acara
Kembali ke proyek hardhat Anda, jalankan:

```
yarn hardhat run scripts/mint-and-list-item.js --network localhost
```

Dan Anda akan melihatnya diperbarui di database

Lesson 16: Hardhat Upgrades

```
git clone https://github.com/PatrickAlphaC/hardhat-upgrades-fcc
cd hardhat-upgrades-fcc
yarn
```

- Typescript

Jika ingin mendapatkan TypeScript dan Anda mengkloning versi javascript, jalankan saja:

```
git checkout typescript
```

- Penggunaan

```
yarn hardhat deploy
```

- Pengujian

```
yarn hardhat test
```

- Cakupan Tes

```
yarn hardhat coverage
```

- Penerapan ke testnet atau mainnet

1. Atur variabel lingkungan

Anda ingin mengatur KOVAN_RPC_URL dan PRIVATE_KEY sebagai variabel lingkungan. Anda dapat menambahkannya ke .envfile, mirip dengan apa yang Anda lihat di .env.example.

- PRIVATE_KEY: Kunci pribadi akun Anda (seperti dari metamask).
CATATAN: UNTUK PENGEMBANGAN, HARAP GUNAKAN KUNCI YANG TIDAK MEMILIKI DANA NYATA YANG TERKAIT DENGANNYA.
 - Anda dapat mempelajari cara mengekspornya di sini .
- KOVAN_RPC_URL: Ini adalah url dari simpul testnet kovan yang sedang Anda kerjakan. Anda bisa mendapatkan pengaturan dengan satu gratis dari Alkimia
- 2. Dapatkan testnet ETH
Buka faucets.chain.link dan dapatkan beberapa tesnet ETH. Anda akan melihat ETH muncul di metamask Anda.
- 3. Menyebarluaskan

```
yarn hardhat deploy --network kovan
```

- Skrip

Setelah menyebarluaskan ke testnet atau jaringan lokal, Anda dapat menjalankan skrip.

```
yarn hardhat run scripts/fund.js
```

atau

```
yarn hardhat run scripts/withdraw.js
```

- Perkirakan gas

Anda dapat memperkirakan berapa biaya bahan bakar dengan menjalankan:

```
yarn hardhat test
```

Dan Anda akan melihat dan mengeluarkan file bernama gas-report.txt

- Verifikasi di etherscan

Jika Anda menerapkan ke testnet atau mainnet, Anda dapat memverifikasinya jika Anda mendapatkan Kunci API dari Etherscan dan menetapkannya sebagai variabel lingkungan bernama ETHERSCAN_API_KEY. Anda dapat memasukkannya ke dalam .envfile Anda seperti yang terlihat di file .env.example.

Dalam keadaan saat ini, jika Anda memiliki set kunci api, itu akan memverifikasi kontrak kovan secara otomatis!

Namun, Anda dapat memverifikasi secara manual dengan:

```
yarn hardhat verify --constructor-args arguments.js  
DEPLOYED_CONTRACT_ADDRESS
```

Lesson 17: Hardhat DAOs

Hardhat DAO

Repo ini dimaksudkan untuk memberi Anda semua pengetahuan yang Anda butuhkan untuk memulai DAO dan melakukan tata kelola. Karena apa gunanya DAO jika Anda tidak dapat membuat keputusan apa pun! Ada 2 jenis utama dalam melakukan pemerintahan.

Fitur	Tata Kelola Rantai	Tata Kelola Hibrida
Biaya Gas	Lebih mahal	lebih murah
Komponen	Hanya blockchain	Oracle atau multisig tepercaya

Struktur tata kelola on-chain yang khas mungkin terlihat seperti:

- Pemungutan suara berbasis ERC20 terjadi pada proyek seperti Tally , tetapi secara hipotesis dapat dilakukan oleh pengguna yang secara manual memanggil fungsi pemungutan suara.
- Siapa pun dapat menjalankan proposal setelah lulus Contoh Senyawa

Tata kelola on-chain bisa jauh lebih mahal, tetapi melibatkan lebih sedikit bagian, dan alatnya masih dikembangkan.

Tata kelola hibrid tipikal dengan oracle mungkin terlihat seperti:

- Pemungutan suara berbasis ERC20 terjadi pada proyek seperti Snapshot
- Oracle seperti Chainlink digunakan untuk mengambil dan mengeksekusi jawaban dengan cara yang terdesentralisasi. (petunjuk petunjuk, seseorang harus membangun ini.)

Tata kelola hibrid tipikal dengan multisig tepercaya mungkin terlihat seperti:

- Pemungutan suara berbasis ERC20 terjadi pada proyek seperti Snapshot
- Multisig gnosis tepercaya digunakan untuk mengeksekusi hasil snapshot. Contoh: Snapsafe

Tata kelola hibrida jauh lebih murah, sama amannya, tetapi alatnya masih terus dikembangkan.

Persyaratan

- git
Anda akan tahu bahwa Anda melakukannya dengan benar jika Anda dapat berlari git --version dan Anda melihat respons seperti git version x.x.x
- Nodejs
Anda akan tahu bahwa Anda telah menginstal nodejs dengan benar jika Anda dapat menjalankannya:
- node --version dan dapatkan output seperti: vx.x.x
- Benang bukannya npm
Anda akan tahu bahwa Anda telah memasang benang dengan benar jika Anda dapat menjalankannya:
- yarn --version dan dapatkan output seperti: x.x.x
Anda mungkin perlu menginstalnya dengan npm atau corepack

```
git clone https://github.com/PatrickAlphaC/hardhat-fund-me-fcc
cd hardhat-fund-me-fcc
yarn
```

1. Kloning repo ini:
git clone https://github.com/PatrickAlphaC/dao-template
cd dao-template
2. Instal dependensi
yarn atau npm i
3. Jalankan test suite (yang juga memiliki semua fungsi)
yarn hardhat test atau npx hardhat test
Jika Anda ingin men-deploy ke testnet:
Tambahkan .envfile dengan konten yang sama dengan .env.example, tetapi diganti dengan variabel Anda.

Anda dapat melakukan semuanya secara manual di jaringan lokal Anda sendiri seperti:

1. Siapkan blockchain lokal
yarn hardhat node
2. Usulkan nilai baru untuk ditambahkan ke kontrak Kotak kami. Di terminal kedua (biarkan blockchain Anda berjalan)
yarn hardhat run scripts/propose.ts --network localhost
3. Berikan suara pada proposal itu
yarn hardhat run scripts/vote.ts --network localhost
4. Antrian & Jalankan proposal!
yarn hardhat run scripts/queue-and-execute.ts --network localhost
5. Anda juga dapat menggunakan wizard kontrak Openzeppelin untuk membuat kontrak lain bekerja dengan variasi kontrak tata kelola ini.

Peta jalan

[] Tambahkan contoh Upgradeability dengan pola proxy UUPS

[] Tambahkan Integrasi Oracle Chainlink dengan contoh Snapsafe

Lihat masalah terbuka untuk daftar lengkap fitur yang diusulkan (dan masalah yang diketahui).

Kontribusilah yang membuat komunitas open source menjadi tempat yang luar biasa untuk belajar, menginspirasi, dan berkreasi. Setiap kontribusi yang Anda buat sangat dihargai .

Jika Anda memiliki saran yang akan membuat ini lebih baik, silakan fork repo dan buat permintaan tarik.

1. Garpu Proyek
2. Buat Cabang Fitur Anda (git checkout -b feature/AmazingFeature)
3. Komit Perubahan Anda (git commit -m 'Add some AmazingFeature')
4. Dorong ke Cabang (git push origin feature/AmazingFeature)
5. Buka Permintaan Tarik

Lesson 18: Security & Auditing

Keamanan Hardhat

Ini adalah bagian dari Kursus Javascript Blockchain/Smart Contract FreeCodeCamp. Bagian kursus ini adalah untuk membantu pengguna memahami keamanan dasar dan beberapa dasar audit.

Audit adalah tinjauan kode yang berfokus pada keamanan untuk mencari masalah dengan kode Anda.

Saat menulis kode yang baik, Anda 100% harus mengikuti ini sebelum mengirim kode Anda ke audit. Tweet dari pakar keamanan legendaris Tinch

- Tambahkan komentar
Ini akan membantu auditor Anda memahami apa yang Anda lakukan.
- Gunakan spesifikasi nat
Dokumentasikan fungsi Anda. DOKUMEN FUNGSI ANDA.
- Uji
Jika Anda tidak memiliki pengujian, dan cakupan pengujian semua fungsi dan baris kode Anda, Anda tidak boleh pergi ke audit. Jika tes Anda tidak lulus, jangan pergi ke audit.
- Bersiaplah untuk berbicara dengan auditor Anda
Semakin banyak komunikasi, semakin baik.
- Bersiaplah untuk memberi mereka banyak waktu.
Mereka benar-benar mencerahkan diri mereka di atas kode Anda.

"Saat ini, ada 0 auditor bagus yang bisa membuat Anda diaudit dalam waktu kurang dari seminggu. Jika seorang auditor mengatakan mereka bisa melakukannya dalam jangka waktu itu, mereka membantu Anda atau tidak." - Patrick Collins , 4 Maret 2022

Proses auditor terlihat seperti ini:

1. Jalankan tes
2. Baca spesifikasi/dokumen
3. Jalankan alat cepat (seperti slither, linter, analisis statis, dll)
4. Analisis Manual
5. Jalankan alat lambat (seperti echidna, manticore, eksekusi simbolis, MythX)
6. Diskusikan (dan ulangi langkah-langkah sesuai kebutuhan)
7. Tulis laporan

Persyaratan

- git
Anda akan tahu bahwa Anda melakukannya dengan benar jika Anda dapat berlari git --version dan Anda melihat respons seperti git version x.x.x
- Nodejs
Anda akan tahu bahwa Anda telah menginstal nodejs dengan benar jika Anda dapat

menjalankan:

- node --version dan dapatkan output seperti:vx.x.x
- Benang bukannya npm

Anda akan tahu bahwa Anda telah memasang benang dengan benar jika Anda dapat menjalankan:

- yarn --version dan dapatkan output seperti:x.x.x
Anda mungkin perlu menginstalnya dengan npm atau corepack
- Buruh pelabuhan

Anda akan tahu bahwa Anda telah menginstal buruh pelabuhan dengan benar jika Anda dapat menjalankan: docker --version dan dapatkan output seperti Docker version xx.xx.xx, build xxxxx

Quickstart

```
git clone https://github.com/PatrickAlphaC/hardhat-security-fcc
cd hardhat-security-fcc
yarn
```

Meluncur

Buka shell buruh pelabuhan:

```
yarn toolbox
```

Lalu lari: slither /src/contracts/
@openzeppelin=/src/node_modules/@openzeppelin
--solc-remaps
--exclude
naming-convention,external-function,low-level-calls

Echidna

Buka shell buruh pelabuhan:

```
yarn toolbox
```

Kemudian, jalankan ini:

```
echidna-test /src/contracts/test/fuzzing/VaultFuzzTest.sol --contract VaultFuzzTest  
--config /src/contracts/test/fuzzing/config.yaml
```

Linting

Untuk memeriksa pemformatan linting/kode:

```
yarn lint
atau
yarn lint:fix
```

Memformat

```
yarn format
```