Course 6: Hardhat Simple Storage

1. Hardhat

   Pada course ini kita akan membuat Hardhat Simple Storage dengan menggunakan beberapa hal yang kita pelajari dari course sebelumnya. Pertama kita akan menginisialisasi package.json dengan perintah *yarn init* dan menginstall hardhat dengan perintah dan *yarn add --dev hardhat* . setelah instalasi selesai kita akan mengganti greeter.sol yang ada di folder contracts menjadi SimpleStorage.sol dengan isi kode sama seperti course sebelumnya yaitu:

```
// I'm a comment!
// SPDX-License-Identifier: MIT
pragma solidity 0.8.8;

// 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;
  }

  function addPerson(string memory _name, uint256 _favoriteNumber) public
{
    people.push(People(_favoriteNumber, _name));
    nameToFavoriteNumber[_name] = _favoriteNumber;
  }
}
```

Pastikan versi solidity di dalam hardhat.config.js sesuai dengan vcersi solidity yang ada dalam kode SimpleStorage.sol agar tidak emndapatkan error.

2. Deploy.js

Pada folder scripts kita akan menemukan file sample-scripts.js, kita akan mengubah nama file tersebut menjadi deploy.js dan menghapus semua isis kode tersebut dan mengganti isi file dengan kode berikut:

```
async function main() {

}

main()
  .then(() => process.exit(0))
  .catch((error) => {
    console.error(error)
    process.exit(1)
  })
```

Kemudian kita akan menginstall prettier plugins menggunakan perintah *yarn add prettier prettier-plugin-solidity*, membuat file baru bernama ".prettierrc"dan mengisi file dengan kode berikut:

```
{
  "tabWidth": 2,
  "useTabs": false,
  "semi": false,
  "singleQuote": false
}
```

Pada file deploy.js kita akan menambahkan beberapa baris kode, yaitu

```
const { ethers, run, network } = require("hardhat")
```

kode di atas akan mengimport ethers, run, dan network dari hardhat yanga kana membantu memudahkan pembuatan simple storage ini. Kemudian dalam fungsi main() kita akan menambahkan kode berikut:

```
const SimpleStorageFactory =
  await ethers.getContractFactory("SimpleStorage")
  console.log("Deploying contract...")
  const simpleStorage = await SimpleStorageFactory.deploy()
  await simpleStorage.deployed()
  console.log(`Deployed contract to: ${simpleStorage.address}`)
```

dengan menggunakan hardhat kita dapat mendeploy smart contract ini tanpa harus mengisi private key atau RPC url, tetapi apabila kita ingin memakai network tertentu dan private key tertentu kita dapat melakukannya dengan meng-edit file hardhat.config.js pada bagian module.exports kiat akan menambahkan

```
defaultNetwork: "hardhat",
networks: {
    hardhat: {},
    rinkeby: {
      url: RINKEBY_RPC_URL,
      accounts: [PRIVATE_KEY],
      chainId: 4,
    },
}
```

Kita akan menambahkan file .env dengan menginstall dotenv dengan perintah *yarn add dotenv* kemudian membuat file .env yang berisi data rcp url dan private key kita. Dan jangan lupa untuk mengimportnya ke hardhat.config.js dengan menambahkan baris ini di bagian paling atas

```
require("dotenv").config()
```

Dan di atas module.exports kita juga akan menambahkan baris ini

```
const RINKEBY_RPC_URL =
  process.env.RINKEBY_RPC_URL

const PRIVATE_KEY =
  process.env.PRIVATE_KEY
```

setelah itu kita akan menambahkan fitur verify kedalam kode kita dengan menggunakan ethescan. Pertama kita harus install etherscan menggunakan perintah *yarn add --dev @nomiclabs/hardhat-etherscan* setelah instalasi selesai kita akan membuat akun di etherscan untuk mendapatkan API key. Saat kita telah mendapatkan API key kita akan menambahkannya dalam file .env kita dengan nama variabel ETHERSCAN_API_KEY dan mengimportnya seperti di atas dengan menambahkan

```
require("@nomiclabs/hardhat-etherscan")
```

dan dalam module.exports kita juga menambahkan

```
etherscan: {
    apiKey: ETHERSCAN_API_KEY,
  },
```

Untuk menggunakan fitur verify ini kita akan meng-edit file deploy.js kita menambahkan fungsi baru dibawah fungsi main() kita bernama “verify”. Dalam fungsi verify ini kita isi dengan kode berikut:

```
const verify = async (contractAddress, args) => {
  console.log("Verifying contract...")
  try {
    await run("verify:verify", {
      address: contractAddress,
      constructorArguments: args,
    })
```

```
  } catch (e) {
    if (e.message.toLowerCase().includes("already verified")) {
      console.log("Already Verified!")
    } else {
      console.log(e)
    }
  }
}
```

Dan menambahkan dalam fungsi main kita

```
const SimpleStorageFactory =
  await ethers.getContractFactory("SimpleStorage")
  console.log("Deploying contract...")
  const simpleStorage = await SimpleStorageFactory.deploy()
  await simpleStorage.deployed()
  console.log(`Deployed contract to: ${simpleStorage.address}`)
 // what happens when we deploy to our hardhat network?

  if (network.config.chainId === 4 && process.env.ETHERSCAN_API_KEY)
 {
    console.log("Waiting for block confirmations...")
    await simpleStorage.deployTransaction.wait(6)
    await verify(simpleStorage.address, [])
  }
  const currentValue = await simpleStorage.retrieve()
  console.log(`Current Value is: ${currentValue}`)

  // Update the current value
  const transactionResponse = await simpleStorage.store(7)
  await transactionResponse.wait(1)
  const updatedValue = await simpleStorage.retrieve()
  console.log(`Updated Value is: ${updatedValue}`)
```

fungsi if akan melakukan verikasi trasaksi kita dan kemudian currentValue akan menampilkan value saat ini dan kemudian akan diperbarui dengan value yang baru oleh updatedValue.

3. (optional) Block-number.js

   Di bagian ini kita akan memcaoba membuat task yang merupakan semacam perintah yang bisa kita panggil tanpa harus memanggil main script kita. Contoh yang kali ini kita coba buat adalah task untuk mendapatkan block number dari network kita, Pertama kita akan membuat folder baru bernama “tasks” dan di dalamnya menambhakn file task baru kita yaitu “block-number.js” yang berisi kode berikut:

```
const { task } = require("hardhat/config")

task("block-number", "Prints the current block number").setAction(
```

```
  async (taskArgs, hre) => {
    const blockNumber = await hre.ethers.provider.getBlockNumber()
    console.log(`Current block number: ${blockNumber}`)
  }
)

module.exports = {}
```

kemudian pada hardhat.config.js kita akan menambahkan import berikut diatas kode

```
require("./tasks/block-number")
```

4. Localhost dan console

   Kita dapat menjalankan virtual blcokchain lokal pada komputer kita mirip seperti genache dengan menggunakan hardhat. Cara nya adalah dengan menggunakan perintah *yarn hardhat node* yang kemudian akan menunjukkan link http localhost yang akan kita tambahkan ke dalam hardhat.config.js dalam module.exports di bagian networks seperti ini:

```
localhost: {
      url: "http://localhost:8545",
      chainId: 31337,
    },
```

   Kita juga dapat memanggil fungsi apapun dalam smart contract kita menggunakan hardhat console. Cara kita untuk mengakses hardhat consol tersebut adalah dengan menggunakan perintah *yarn hardhat console --network {network names}* yang akan memasukkan kita ke dalam shell hardhat. Dalam shell hardhat ini kita dalam memanggil fungsi apapun dari project kita, hal ini digunakan untuk testing apakah suatu fungsi berjalan edngan benar tanpa menjalan harus menjalankan seluruh file .js.

5. Test-Deploy.js dan gasReporter

   Untuk meningkatkan keamanan dari smart contract kita, kita akan membuat test-deploy.js pada folder test yang akan menggantikan sample-test.js. file test ini berguna untuk kita melakukan testing pada contract kita agar setiap fungsi berjalan dengan benar saat kita mempublish kode kita ke publik. Dalam file test-deploy.js ini kita akan isi dengan kumpulan kode berikut:

```
const { ethers } = require("hardhat")
const { expect, assert } = require("chai")

// describe("SimpleStorage", () => {})
describe("SimpleStorage", function () {
  // let simpleStorageFactory
  // let simpleStorage
  let simpleStorageFactory, simpleStorage
  beforeEach(async function () {
    simpleStorageFactory = await
ethers.getContractFactory("SimpleStorage")
```

```
    simpleStorage = await simpleStorageFactory.deploy()
  })

  it("Should start with a favorite number of 0", async function ()
    { const currentValue = await simpleStorage.retrieve()
    const expectedValue = "0"
    // assert
    // expect
    assert.equal(currentValue.toString(), expectedValue)
    // expect(currentValue.toString()).to.equal(expectedValue)
  })
  it("Should update when we call store", async function () {
    const expectedValue = "7"
    const transactionResponse = await
simpleStorage.store(expectedValue)
    await transactionResponse.wait(1)

    const currentValue = await simpleStorage.retrieve()
    assert.equal(currentValue.toString(), expectedValue)
  })

  // Extra - this is not in the video
  it("Should work correctly with the people struct and array",
async function () {
    const expectedPersonName = "Patrick"
    const expectedFavoriteNumber = "16"
    const transactionResponse = await simpleStorage.addPerson(
      expectedPersonName,
      expectedFavoriteNumber
    )
    await transactionResponse.wait(1)
    const { favoriteNumber, name } = await simpleStorage.people(0)
    // We could also do it like this
    // const person = await simpleStorage.people(0)
    // const favNumber = person.favoriteNumber
    // const pName = person.name

    assert.equal(name, expectedPersonName)
    assert.equal(favoriteNumber, expectedFavoriteNumber)
  })
})
```

Kita juga melakukan testing mengenai gas price dari fungsi yang kita pakai menggunakan gas reporter. Kita akan menginstall gas reporter dengan perintah *yarn add hardhat-gas-reporter --dev* . kemudian setelah selesai kita akan mengimportnya pada hardhat.config.js kita seperti ini

```
require("hardhat-gas-reporter")
```

kita juga akan menambahkan export ke module.exports seperti ini

```
gasReporter: {
    enabled: true,
    currency: "USD",
    outputFile: "gas-report.txt",
    noColors: true,
    coinmarketcap: COINMARKETCAP_API_KEY,
  },
```

Untuk mendapatkan COINMARKETCAP_API_KEY kita akan melakukan hal yang sama seperti etherscan. Masuk ke website CoinMarketCap dan registrasi akun baru, setelah selesai registrasi carilah API key dari akun kita dan kita dapat menambahkannya ke .env file kita dan memanggilnya pada hardhat.config.js menggunakan kode berikut:

```
const ETHERSCAN_API_KEY = process.env.ETHERSCAN_API_KEY || ""
```

terakhir untuk tambahan keamanan kit dapat menggunakan solidity coverage. Ini adalah module yang akan melakukan scan pada kode kita dan mencari kelemahan dari keamanan dari kode kita. Kita dapat menggunakan solidity coverage dengan menginstall nya dengan perintah *yarn add --dev solidity-coverage* dan menambahkan importnya pada hardhat.config.js kita seperti ini

```
require("solidity-coverage")
```

kita dapat melakukan coverage dengan perintah *yarn hardhat coverage* yang kemudian akan memberikan kita hasil coverage kode kita dalam file baru bernama “coverage.json”.