Setup and Overview

In this article, you'll learn about Hardhat: a development framework to create, test, and deploy smart contracts to Ethereum and other supported EVM chains.

Objectives

By the end of this lesson, you should be able to:

Install and create a new Hardhat project with Typescript support
Describe the organization and folder structure of a Hardhat project
List the use and properties of hardhat.config.ts

Overview

Hardhat is a development environment that allows you to develop and test Solidity on your local machine. It includes debugging and unit testing tools, and has an ecosystem of third-party-developed plugins that ease development and deployment.

Among other things, these plugins can help you deploy contracts, see the size of your compiled byte-code, and even see unit test coverage.

Installing Hardhat and creating a new project

As a pre-requisite to start developing smart contracts with Hardhat, Node.js must be installed.

You can then simply type `npx hardhat init`, which provides a set of options to bootstrap a Hardhat project:

```
888    888                   888 888            888
888    888                   888 888            888
888    888                   888 888            888
8888888888  8888b.  888d888 .d88888 88888b.   8888b.  888888
888    888     "88b 888P"  d88" 888 888 "88b     "88b 888
888    888 .d888888 888    888  888 888  888 .d888888 888
888    888 888  888 888    Y88b 888 888  888 888  888 Y88b.
888    888 "Y888888 888     "Y88888 888  888 "Y888888  "Y888
👷 Welcome to Hardhat v2.11.2 👷
? What do you want to do? …
❯ Create a JavaScript project
  Create a TypeScript project
  Create an empty hardhat.config.js
  Quit
```

You are encouraged to select Create a TypeScript project, since it provides you with some benefits such as static typing that can reduce the number of errors during development.

You can then enter 'yes' for the remaining options, which include installing the `@nomicfoundation/hardhat-toolbox` package that contains some of the most used Hardhat plugins.

```
✓ What do you want to do? · Create a TypeScript project
✓ Hardhat project root: · {any location}
✓ Do you want to add a .gitignore? (Y/n) · y
✓ Do you want to install this sample project's dependencies with npm (hardhat @nomicfoundation/hardhat-toolb
```

Anatomy of a Hardhat project

After you complete the previous step, the folder structure looks like the following:

```
contracts # contracts will go here
hardhat.config.ts # configuration file for hardhat
node_modules # node.js package folder
package-lock.json # node.js package lock file
package.json # node.js package file
scripts # place the scripts here
test # place the tests here
tsconfig.json # typescript configuration file
```

It is also common to save hardhat tasks in a `task` folder.

It is important to mention that all these paths are fully configurable in the `hardhat.config.ts` file. You can specify a different folder for the contracts, such as `src`.

Configuration

You can configure the Hardhat environment in the `hardhat.config.ts` file.

Since the project uses Typescript, you have the benefit of using static typing.

The following is the default configuration:

```
import { HardhatUserConfig } from 'hardhat/config';
import '@nomicfoundation/hardhat-toolbox';
const config: HardhatUserConfig = {
  solidity: '0.8.17',
};
export default config;
```

You can configure aspects such as:

```
default network
networks
solidity
paths
mocha
```

For example:

```
import { HardhatUserConfig } from 'hardhat/config';
import '@nomicfoundation/hardhat-toolbox';
const config: HardhatUserConfig = {
  defaultNetwork: 'base',
  networks: {
    base_sepolia: {
      url: 'https://sepolia.base.org',
      accounts: ['<private key 1>'],
    },
    sepolia: {
      url: 'https://sepolia.infura.io/v3/<key>',
      accounts: ['<private key 1>', '<private key 2>'],
    },
  },
```

```
  solidity: {
    version: '0.8.17',
    settings: {
      optimizer: {
        enabled: true,
        runs: 200,
      },
    },
  },
  paths: {
    sources: './contracts',
    tests: './test',
    cache: './cache',
    artifacts: './artifacts',
  },
};
export default config;
```

Compiling smart contracts

At this point, you should have a Hardhat project up and running to start developing smart contracts. You may notice Hardhat includes a sample contract called `Lock.sol`.

To run your first command, enter `npx hardhat compile`, which compiles the smart contracts and generates the correct artifacts that includes the bytecode and ABI.

After running the `npx hardhat compile` command, you should see a new folder named artifacts. This folder contains each contract name as a folder and a `{ContractName}.json` file.