Smart Contracts Exercise 01: Hello, Blockchain World!

Welcome to the first smart contracts exercise! In this first exercise, you will become familiar with the basics of smart contract development. The goal is to create a simple smart contract. You will then compile, test, and deploy this smart contract in the local network environment, and subsequently deploy it to the live blockchain.

1 Task: Set Up Hardhat Environment

In this task, you will set up the Hardhat development environment. Hardhat is a development environment for Ethereum software. It provides a suite of tools for editing, compiling, debugging, and deploying your smart contracts and decentralized applications. You have two options for working with this exercise. Using docker container or local installation. Choose the one that best fits your preferences.

1.1 Using Docker with VS Code

This option uses Docker to create a development environment with all the necessary tools and dependencies pre-installed.

Prerequisites:

- **Docker** - A platform for developing, shipping, and running applications in containers.
- **Visual Studio Code** - A lightweight but powerful source code editor.
- **Dev Containers** - An extension to VS Code that lets you use a Docker container as a full-featured development environment.

Setting Up the Project:

1. Visit the following GitLab repository and clone it to your local machine.
2. Open the repository folder in VS Code.
3. When prompted, click "Reopen in Container" or use the command palette (F1) and run `Dev Containers: Reopen in Container`.

1.2 Local Setup

If you prefer working directly on your machine without Docker, you can set up the development environment locally. Before setting up Hardhat, ensure that you have the following installed on your system:

Prerequisites

- **Node.js** - https://nodejs.org/en/ - An open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser.
- **NPM**: Node Package Manager, which comes with Node.js.

Open your terminal and run the following commands to verify the installations:

```
$ node -v
$ npm -v
```

Both commands should return the installed version numbers of Node.js and NPM respectively. Node.js provides the runtime environment required to execute JavaScript-based tools like Hardhat, while NPM is used to manage the packages and dependencies needed for development.

- **Tip 1:** If you are using Windows, we strongly recommend using Windows Subsystem for Linux (WSL) to follow this guide. For more information, refer to the official documentation.
- **Tip 2:** If you are using Visual Studio Code, consider installing the Visual Studio Code Solidity Extension. This extension helps your development process by providing features like syntax highlighting, code completion, etc.

1.3 Creating a New Hardhat Project

Create an empty working directory and then run the following commands to initialize a Hardhat project:

```
$ npm init -y # Initialize an npm project in the directory.
$ npm install --save-dev hardhat # Install Hardhat in the directory.
$ npx hardhat init # Initialize a Hardhat project.
```

Select `Create an empty hardhat.config.js` with your keyboard and hit enter.

2 Task: Writing Your First Smart Contract

Start by creating a new directory inside your project called `contracts` and create a file inside the directory called `Greeter.sol`. Paste the code below into the file and take a minute to read the code.

```
// File: contracts/Greeter.sol

// SPDX-License-Identifier: MIT
pragma solidity ^0.8.28; // Specify the Solidity compiler version

/**
 * @title Greeter
 * @dev A simple smart contract that stores a greeting message.
 */
contract Greeter {
    string private greeting; // State variable to store the greeting message

    /**
    * @dev Constructor that sets the initial greeting message upon deployment.
     * @param _greeting The greeting message to be stored.
     */
    constructor(string memory _greeting) {
        greeting = _greeting;
    }

    /**
     * @dev Function to retrieve the greeting message.
     * @return The current greeting stored in the contract.
     */
    function greet() public view returns (string memory) {
        return greeting;
    }
}
```

The Greeter contract is a simple Solidity smart contract that stores a greeting message, initializes it during deployment, and allows users to retrieve it via a public function. To compile the contract, run `npx hardhat compile` in your terminal.

```
$ npx hardhat compile
Compiled 1 Solidity file successfully (evm target: paris).
```

Hardhat compiled your Solidity smart contract and generated corresponding artifacts—including the contract's ABI, bytecode, and related metadata—which are stored in the `artifacts` folder.

3 Task: Test your Smart Contract with Local Hardhat Network

3.1 Set Up Hardhat-Toolbox Plugin

In this task, you will write and execute a simple test case for the `Greeter` contract using Hardhat's local network. For this task, we will need the `@nomicfoundation/hardhat-toolbox` plugin. It integrates testing libraries, Ethers.js, and other deployment utilities. Run the following command in the directory to install the plugin:

```
$ npm install --save-dev @nomicfoundation/hardhat-toolbox
```

To include the plugin in your Hardhat project, add the following to your `hardhat.config.js` file in the project directory so that it will look like this:

```
// File: hardhat.config.js

require("@nomicfoundation/hardhat-toolbox");

/** @type import('hardhat/config').HardhatUserConfig */
module.exports = {
  solidity: "0.8.28",
};
```

For more information about plugins and how to test contracts in Hardhat, visit: Hardhat documentation.

3.2 Writing a Simple Test

Create a new directory named `test` in your project root and add a file called `Greeter.js` with the following content:

```
// File: test/Greeter.js

// Import the 'expect' function from Chai for assertions
const { expect } = require("chai");

// Test suite for the Greeter contract
describe("Greeter contract says Hello, Blockchain World!", function () {

  // Test to ensure the initial greeting is set correctly upon deployment
  it("Deployment should set the initial greeting correctly.", async function () {

    // Define the initial greeting message
    const initialGreeting = "Hello, Blockchain World!";

    // Deploy the Greeter contract with the initial greeting
    const greeter = await ethers.deployContract("Greeter", [initialGreeting]);

    // Wait for the deployment to complete
    await greeter.waitForDeployment();

    // Retrieve the stored greeting from the contract
    const greeting = await greeter.greet();

    // Verify that the retrieved greeting matches the initial greeting
    expect(greeting).to.equal(initialGreeting);
  });
});
```

3.3 Running the Test

Execute the test on Hardhat's local network by running the following commands in your terminal:

```
$ npx hardhat test
```

Congratulations! You wrote, compiled, and tested your first smart contract!

4 Task: Deploying to a Live Network

Once you have programmed and tested your dApp, you want to deploy it to a public blockchain so that others can access it. For the purposes of our exercise, we will not use the Ethereum mainnet because we would have to pay with real money, but instead use a live testnet. A testnet mimics real-world scenarios without risking our own money. Ethereum has several testnets; for our purposes, we will choose the Sepolia testnet. Deploying to a testnet is the same as deploying to mainnet at the software level. The only difference is the network you connect to.

4.1 Prerequisites

In order to finish this task, you will need the following tools:

- **MetaMask**: A popular Ethereum wallet that allows you to interact with the Ethereum blockchain. You can download the MetaMask extension for your browser from the official website and set it up. But you can also use other wallets or simply create your own private-public key pair.
- **Infura API Key**: Infura provides access to Ethereum nodes without the need to run your own. Sign up at Infura to obtain an API key.
- **Sepolia Faucet**: Acquire Sepolia test Ether (ETH) from a faucet to fund your deployment. Some reliable faucets include:
  - Google Cloud Web3
  - Metamask Sepolia Faucet
  - Alchemy Sepolia Faucet

4.2 Configuring Hardhat for Sepolia Deployment

To deploy your smart contract to the Sepolia testnet, you need to configure Hardhat with the network details and your wallet credentials.

Storing Sensitive Information

It's crucial to keep sensitive information like your private key and Infura API key secure. We recommend using environment variables to manage these credentials only for the purpose of this exercise. A Hardhat project can use configuration variables for user-specific values or for data that shouldn't be included in the code repository. These variables are set via tasks in the vars scope and can be retrieved in the config using the vars object.

- Set the INFURA_API_KEY

```
$ npx hardhat vars set INFURA_API_KEY
Enter value: ********************************
```

- Set the SEPOLIA_PRIVATE_KEY

```
$ npx hardhat vars set SEPOLIA_PRIVATE_KEY
Enter value: ********************************
```

- **Warning 1**: Configuration variables are stored in plain text on your disk. Avoid using this feature for data you wouldn't normally save in an unencrypted file. Run `npx hardhat vars path` to find the storage's file location.
- **Warning 2**: Never store your private keys as plain text, even in your `.env` file. Use secure key management tools. For the purpose of our exercise, we will use environment variables. **Never use your private key associated with real money in plain text!**

Updating `hardhat.config.js`

Modify your `hardhat.config.js` file to include the Sepolia network configuration:

```
// File: hardhat.config.js

require("@nomicfoundation/hardhat-toolbox");

const INFURA_API_KEY = vars.get("INFURA_API_KEY");
const SEPOLIA_PRIVATE_KEY = vars.get("SEPOLIA_PRIVATE_KEY");

/** @type import('hardhat/config').HardhatUserConfig */
module.exports = {
  solidity: "0.8.28",
  networks: {
    sepolia: {
      url: `https://sepolia.infura.io/v3/${INFURA_API_KEY}`,
      accounts: [`0x${SEPOLIA_PRIVATE_KEY}`],
    },
  },
};
```

This configuration tells Hardhat how to connect to the Sepolia testnet using your Infura API key and deploy contracts using your wallet's private key.

4.3 Deploying the Smart Contract to Sepolia

With the configuration in place, you're ready to deploy your smart contract to the Sepolia testnet.

4.3.1 Creating a Deployment Script

- 1. Create a `scripts` Directory: In your project root, create a new directory named `scripts`.
- 2. Add a Deployment Script: Inside the `scripts` directory, create a file named `deploy.js` and add the following content:

```
// File: scripts/deploy.js

const hre = require("hardhat");

async function main() {
    // Set the initial greeting message
    const initialGreeting = "Hello, Blockchain World!";

    // Deploy the Greeter contract with the initial greeting
    const greeter = await ethers.deployContract("Greeter", [initialGreeting]);
    console.log(`Greeter contract deployed to: ${greeter.target}`);

    // Wait for the deployment to complete
    await greeter.waitForDeployment();

    // Retrieve the stored greeting from the contract
    const greeting = await greeter.greet();
    console.log(`Contract greeting: ${greeting}`);
}

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

4.3.2 Executing the Deployment

Run the deployment script using Hardhat with the Sepolia network specified:

```
$ npx hardhat run scripts/deploy.js --network sepolia
```

Expected Output:

```
Greeter contract deployed to: <ContractAddress>
Contract greeting: Hello, Blockchain World!
```

You can verify the deployment by visiting the Sepolia Etherscan explorer and searching for your contract address: https://sepolia.etherscan.io/address/<ContractAddress>. Search also for your account address and see your interactions with the deployed contract. Hardhat also includes Hardhat network, a local Ethereum network node for development. It enables you to deploy contracts, run tests, and debug code, all within your local environment. We already used it during running our test. To use it, open a separate terminal and run `npx hardhat node` in the terminal. To deploy the contract, run `npx hardhat run scripts/deploy.js --network hardhat` in another terminal. See Hardhat network for more information.

4.4 Interacting with Your Deployed Contract

Now that your contract is live on the Sepolia testnet, you can interact with it using various tools:

- **Etherscan**: View contract details, read functions, and execute transactions directly from the Etherscan interface.
- **Web3 Interfaces**: Integrate your contract with frontend applications using libraries like `ethers.js` or `web3.js`.
- **Hardhat Tasks**: Write scripts or use the Hardhat console to interact programmatically with your contract.

Tip: If you run the deployment script without specifying the `--network` parameter, it will deploy to the local Hardhat network.

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

4.5 Further Reading

For more detailed information, refer to the following resources:

- Solidity Documentation
- Hardhat Documentation
- Solidity by Example
- Ethers.js Documentation (for scripting)
- Chai Assertion Library

Congratulations! You have successfully deployed your first smart contract to the live blockchain network! Stay tuned for the upcoming exercises!