Deploy a Smart Contract Using Hardhat

In this tutorial, you'll be guided through setting up a Hardhat project and deploying a Hedera smart contract to the **Hedera Testnet** using the **Hashio** JSON-RPC instance.

Hardhat is a development environment for Ethereum smart contracts. It consists of different components for editing, compiling, debugging, and deploying your smart contracts and dApps, all working together to create a complete development environment. By the end of this tutorial, you'll have learned how to deploy smart contracts using Hardhat on the Hedera Testnet**.**

Prerequisites

- Basic understanding of smart contracts.
- Basic understanding of Node.js or Javascript.
- Basic understanding of the Hedera JSON RPC Relay.
- Basic understanding of Hardhat Ethereum Development Tool.

Hardhat Project Setup

To make the setup process simple, you'll use a pre-configured Hardhat project from the `hedera-hardhat-example-project` repository.

Open a terminal window and navigate to your preferred directory where your Hardhat project will live. Run the following command to clone the repo and install dependencies to your local machine:

```
git clone https://github.com/hashgraph/hedera-hardhat-example-project.git
cd hedera-hardhat-example-project
npm install
```

The `dotenv` package is used to manage environment variables in a separate `.env` file, which is loaded at runtime. This helps protect sensitive information like your private keys and API secrets, but it's still best practice to add `.env` to your `.gitignore` to prevent you from pushing your credentials to GitHub.

Project Configuration

In this step, you will update and configure the Hardhat configuration file that defines tasks, stores Hedera account private key information, and Hashio Testnet RPC URL. First, rename the `.env.example` file to `.env`. and update the `.env` and `hardhat.config.js` files with the following code.

But first, in order to deploy the contract to the *Hedera Testnet*, you will need to get a testnet account and key. To get a testnet account, create an *ECDSA* account using the **Hedera Developer Portal**. Once you have your ECDSA account and HEX encoded private key, add the private key to the `TESTNET_OPERATOR_PRIVATE_KEY` variable in the `.env` file.

> Create your testnet account 🛠️

Environment Variables

The `.env` file defines environment variables used in the Hardhat configuration file. The `TESTNET_OPERATOR_PRIVATE_KEY` variable contains the *ECDSA* ***Hex Encoded Private Key*** for the Hedera Testnet account used in the `testnet` network Hardhat configuration.

The `TESTNET_ENDPOINT` variable contains the HashIO Testnet endpoint URL. This is the JSON-RPC instance that will submit the transactions to the Hedera test network to test,

create and deploy your smart contract.

```
# operator/receiver keys referenced in the hardhat.config account variable
TESTNET_OPERATOR_PRIVATE_KEY=0xb46751179bc8aa9e129d34463e46cd924055112eb30b31637

# testnet endpoint referenced in the hardhat.config url variable
TESTNET_ENDPOINT='https://testnet.hashio.io/api'
```

Hardhat Configuration

The `hardhat.config.js` file defines tasks for Hardhat, including `show-balance`, `transfer-hbars`, `deploy-contract`, `contract-view-call`, and `contract-call`. It exports a configuration object that includes the Solidity version and settings, default network, and network settings for the `testnet` network.

The `url` property is set to the `TESTNET_ENDPOINT` environment variable, and `accounts` to an array containing the testnet private key imported from the `.env` file.

```
require("@nomicfoundation/hardhat-toolbox");
require("@nomicfoundation/hardhat-chai-matchers");
require("@nomiclabs/hardhat-ethers");
//import dotenv library to access environment variables stored in .env file
require("dotenv").config();

//define hardhat task here, which can be accessed in our test file (test/rpc.js)
task("show-balance", async () => {
  const showBalance = require("./scripts/showBalance");
  return showBalance();
});

task("deploy-contract", async () => {
  const deployContract = require("./scripts/deployContract");
  return deployContract();
});

task("contract-view-call", async (taskArgs) => {
  const contractViewCall = require("./scripts/contractViewCall");
  return contractViewCall(taskArgs.contractAddress);
});

task("contract-call", async (taskArgs) => {
  const contractCall = require("./scripts/contractCall");
  return contractCall(taskArgs.contractAddress, taskArgs.msg);
});

/** @type import('hardhat/config').HardhatUserConfig */
module.exports = {
  mocha: {
    timeout: 3600000,
  },
  solidity: {
    version: "0.8.9",
    settings: {
      optimizer: {
        enabled: true,
        runs: 500,
      },
    },
  },
  //this specifies which network should be used when running Hardhat tasks
  defaultNetwork: "testnet",
  networks: {
    testnet: {
      //HashIO testnet endpoint from the TESTNET_ENDPOINT variable in the projec
      url: process.env.TESTNET_ENDPOINT,
      //the Hedera testnet account ECDSA private
      //the public address for the account is derived from the private key
```

```
      //the public address for the account is derived from the private key
      accounts: [
        process.env.TESTNET_OPERATOR_PRIVATE_KEY,
      ],
    },
  },
};
```

Project Contents

In this step, you'll look at the descriptions of the Hardhat project contents. For more information regarding Hardhat projects, check out the Hardhat docs. If you do not need to review the project contents you can skip to "Test and Deploy."

contracts/

The `contracts/` folder contains the source file for the Greeter smart contract.

Let's review the `Greeter.sol` smart contract in the `hedera-example-hardhat-project/contracts` folder. At the top of the file, the `SPDX-License-Identifier` defines the license, in this case, the MIT license. The `pragma solidity ^0.8.9;` line specifies the Solidity compiler version to use. These two lines are crucial for proper licensing and compatibility.

```
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

contract Greeter {
    string private greeting;

    event GreetingSet(string greeting);

    //This constructor assigns the initial greeting and emit GreetingSet event
    constructor(string memory _greeting) {
        greeting = _greeting;

        emit GreetingSet(_greeting);
    }

    //This function returns the current value stored in the greeting variable
    function greet() public view returns (string memory) {
        return greeting;
    }

    //This function sets the new greeting msg from the one passed down as parame
    function setGreeting(string memory _greeting) public {
        greeting = _greeting;

        emit GreetingSet(_greeting);
    }
}
```

> *NOTE: The pragma solidity line must match the version of Solidity defined in the module exports of your `hardhat.config.js` file.*

scripts/

The `scripts/` folder contains the automation scripts for the test file. This project contains 4 scripts.

The `scripts/` folder contains test scripts for locally testing a smart contract before deploying it. Please read the comments to help you understand the code and its purpose:

contractCall.js | contractViewCall.js | deployContract.js | showBalance.js

Calls the `setGreeting` function from the Greeter contract and sets the greeter message to "Greeter."\

```
const { ethers } = require('hardhat');

//This function accepts two parameters - address and msg
//Retrieves the contract from the address and set new greeting
module.exports = async (address, msg) => {

  //Assign the first signer, which comes from the first privateKey from our
  const wallet = (await ethers.getSigners())[0];

  //Assign the greeter contract object in a variable, this is used for alrea
  //name of contract as first parameter
  //address of our contract
  //wallet/signer used for signing the contract calls/transactions with this
  const greeter = await ethers.getContractAt('Greeter', address, wallet);

  //using the greeter object(which is our contract) we can call functions fr
  const updateTx = await greeter.setGreeting(msg);

  console.log(`Updated call result: ${msg}`);

  return updateTx;
};
```

test/

The `test/` folder contains the test files for the project.

The `rpc.js` file is located in the `test` folder of the `hedera-example-hardhat-project` project and references the Hardhat tasks that are defined in the hardhat.config file. When the command `npx hardhat test` is run, the program executes the `rpc.js` file.

```
const hre = require("hardhat");
const { expect } = require("chai");

describe("RPC", function () {
  let contractAddress;
  let signers;

  before(async function () {
    signers = await hre.ethers.getSigners();
  });

  it("should be able to get the account balance", async function () {
    const balance = await hre.run("show-balance");
    expect(Number(balance)).to.be.greaterThan(0);
  });

  it("should be able to deploy a contract", async function () {
    contractAddress = await hre.run("deploy-contract");
    expect(contractAddress).to.not.be.null;
  });

  it("should be able to make a contract view call", async function () {
    const res = await hre.run("contract-view-call", { contractAddress });
    expect(res).to.be.equal("initial_msg");
  });

  it("should be able to make a contract call", async function () {
    const msg = "updated_msg";
    await hre.run("contract-call", { contractAddress, msg });
    const res = await hre.run("contract-view-call", { contractAddress });
    expect(res).to.be.equal(msg);
  });
});
```

.env

A file that stores your environment variables like your accounts, private keys, and references to Hedera network (see previous step).

hardhat.config.js

The Hardhat project configuration file. This file includes information about the Hedera network RPC URLs, accounts, and tasks defined (see previous step).

Test and Deploy

Now that you have your project set up and configured, let's deploy the `Greeter.sol` smart contract to the Hedera Testnet using Hashio. Hashio is an instance of the Hedera JSON-RPC relay hosted by Swirlds Labs and provides convenient access to the Hedera network for transactions and data querying. You can use any JSON-RPC instance supported by the community.

Run the following command in your terminal to run the `hedera-hardhat-example-project/test/rpc.js` test file on the Hedera testnet.

```
npx hardhat test
```

Tests should pass with "**4 passing**" returned to the console. Otherwise, an error message will appear indicating the issue.

```
> console check ✅
```

Lastly, run the following command to deploy the contract to the Hedera Testnet:

```
npx hardhat deploy-contract
```

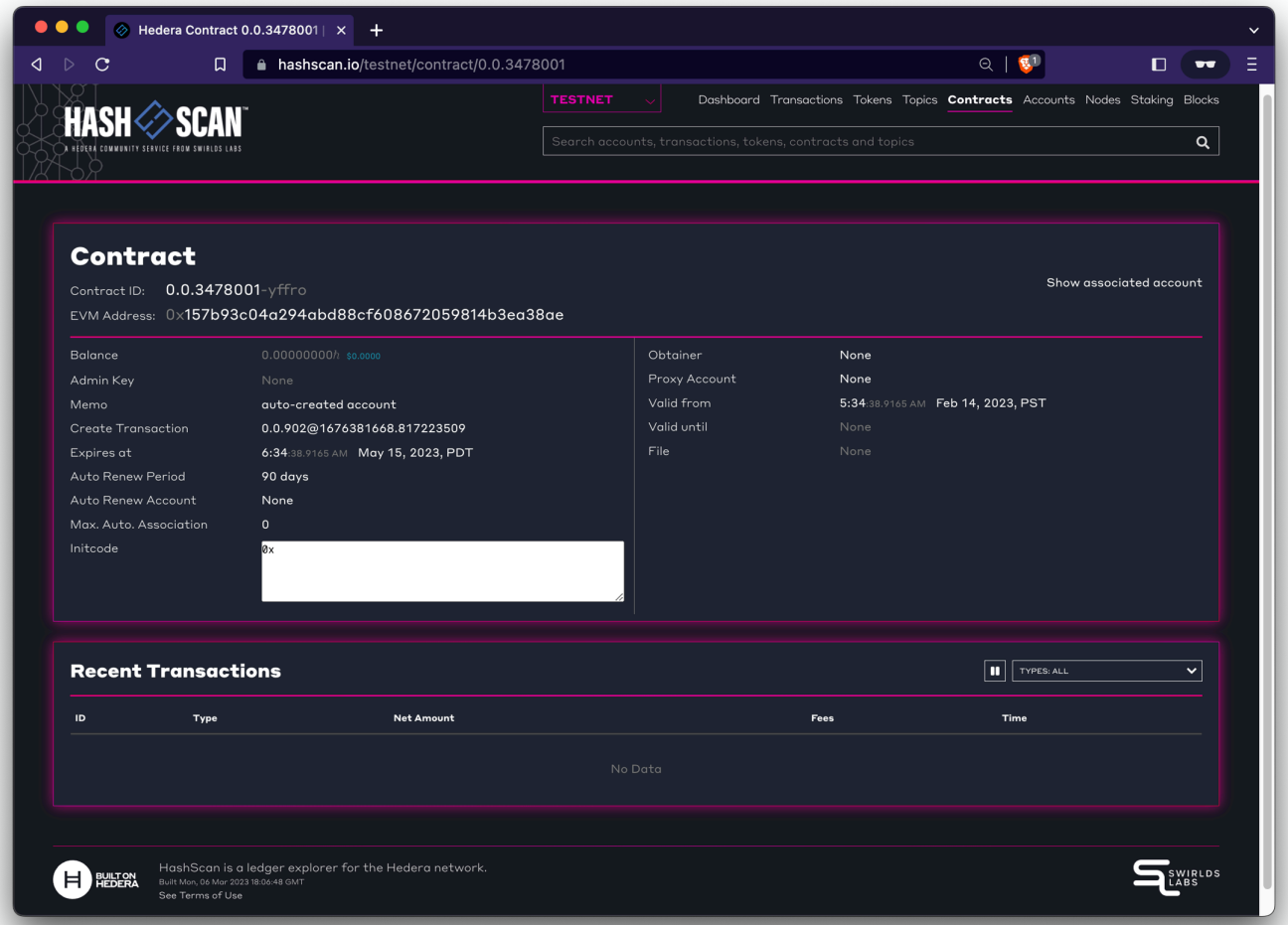
```
> console check ✅
```

You've successfully deployed your contract to the Hedera Testnet! You can view the contract you deployed by searching the smart contract *public* address in a supported Hedera Network Explorer. For this example, we will use the HashScan Network Explorer. Copy and paste your deployed `Greeter.sol` public contract address into the HashScan search bar.

The Network Explorer will return the information about the contract created and deployed to the Hedera Testnet. The "EVM Address" field is the public address of the contract that

was returned to you in your terminal. The terminal returned the public address with the "0x" hex encoding appended to the public address. You will also notice a contract ID in `0.0.contractNumber` (`0.0.3478001`) format. This is the *contract ID* used to reference the contract entity in the Hedera Network.

> ***Note:*** *At the top of the explorer page, remember to switch the network to* ***TESTNET*** *before you search for the contract.*



Hashscan transaction

Congratulations! 🎉 You have successfully learned how to deploy a smart contract using Hardhat. Feel free to reach out if you have any questions:

Writer: Krystal, Technical Writer

GitHub | Twitter

Editor: Simi, Sr. Software Manager

GitHub | LinkedIn

Previous
Deploy By Leveraging Ethereum Developer Tools On Hedera

Next
Deploy a Subgraph Using The Graph and JSON-RPC

Last updated 1 year ago