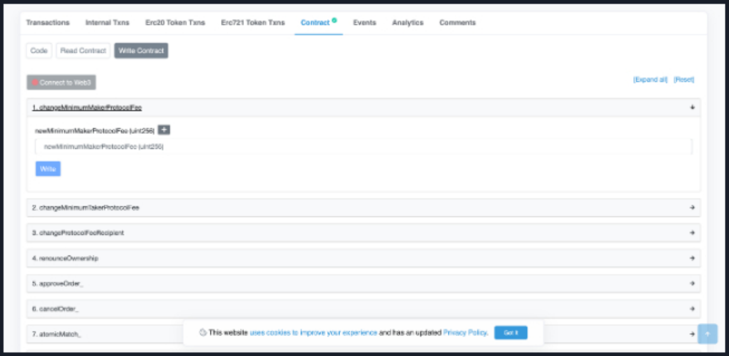
Lesson Type: Practical | Estimated Time: 30-60 minutes | Current Score: 0%

Verify your Smart Contracts on Etherscan

If you open **this** etherscan link, you can see that you can interact with this smart contract's functions directly through etherscan, similar to how you used to do it on Remix.



This website uses cookies to improve your experience and has an updated Privacy Policy.

Are you wondering why that doesn't happen for your contract?

The reason is that the contract mentioned above is verified on etherscan while yours is not. So lets learn why and how to verify contracts on etherscan 🚀

Why?

Verifying contracts is important because it ensures that the code is exactly what was deployed onto the blockchain. It also allows the public to read and audit your smart contract code and is a good signal for establishing trust for your dApp. It also gives you an UI interface to interact with the contracts similar to Remix, where you can call all the functions that exist directly through Etherscan.

Hardhat Plugin

With all that in mind though, it is important to note that verifying code on Etherscan manually can get quite tricky. You need to make sure that you not only verify your main contract but also the contracts that you are inheriting or using along with your main contract. If you deployed your contract for testing and verified it already with the slightest of changes to your contract you will have to again go through the manual process which gets tedious over time.

Build

Lets now learn how we can leverage hardhat for verifying smart contracts with only some lines of code.

Lets goo 🚀🚀🚀

Write some code to verify

Open up a terminal and execute these commands

```
mkdir etherscan-verification
```

```
cd etherscan-verification
```

You will now need to set up your Hardhat project

```
npm init --yes
npm install --save-dev hardhat
```

Make sure you select `Create a Javascript project` and then follow the steps in your terminal.

Now create a new file inside the `contracts` directory called `Verify.sol`.

```
//SPDX-License-Identifier: Unlicensed
pragma solidity ^0.8.4;

contract Verify {
    string private greeting;

    constructor() {
    }

    function hello(bool sayHello) public pure returns (string memory) {
        if(sayHello) {
            return "hello";
        }
        return "";
    }
}
```

We will install `dotenv` package to be able to import the env file and use it in our config. Open up a terminal pointing at `etherscan-verification` directory and execute this command

```
npm install dotenv
```

Today, instead of Goerli, we will start using the Mumbai network. Mumbai is a testnet for the Polygon network. Consider this your first introduction to building on Polygon. (Don't worry, it's very similar to building on Ethereum).

Create a `.env` file in the `etherscan-verification` folder and add the following lines. Follow the instructions below:

Go to **Quicknode** and sign up for an account. If you already have an account, log in. Quicknode is a node provider that lets you connect to various different blockchains. We will be using it to deploy our contract through Hardhat. After creating an account, `Create an endpoint` on Quicknode, select `Polygon`, and then select the `Mumbai` network. Click `Continue` in the bottom right and then click on `Create Endpoint`. Copy the link given to you in `HTTP Provider` and add it to the `.env` file below for `QUICKNODE_HTTP_URL`.

> NOTE: If you previously set up a Goerli Endpoint on Quicknode, you need to delete it and create a new one for Mumbai. If you already have a Mumbai endpoint, you can keep using that one.

To get your private key, you need to export it from Metamask. Open Metamask, click on the three dots, click on `Account Details` and then `Export Private Key`. MAKE SURE YOU ARE USING A TEST ACCOUNT THAT DOES NOT HAVE MAINNET FUNDS FOR THIS. Add this Private Key below in your `.env` file for `PRIVATE_KEY` variable.

Also, make sure you have some Mumbai `MATIC` tokens to work with. If you don't know how to get them, follow **this guide by ThirdWeb**.

Lastly, similar to Etherscan, the Polygon network has Polygonscan. Both block explorers are developed by the same team and work basically exactly the same way. To automate contract verification on Polygonscan through Hardhat, we will need an API Key for Polygonscan. Go to **PolygonScan** and sign up. On the Account Overview page, click on `API Keys`, add a new API Key, and copy the `API Key Token`. Put this in `POLYGONSCAN_KEY` below.

```
QUICKNODE_HTTP_URL="add-quicknode-http-provider-url-here"

PRIVATE_KEY="add-the-private-key-here"

POLYGONSCAN_KEY="polygonscan-api-key-token-here"
```

-

Lets deploy the contract to `mumbai` network. Create a new file, or replace the existing default one, named `deploy.js` under the `scripts` folder. Notice how we are using code to verify the contract.

```
const { ethers } = require("hardhat");
require("dotenv").config({ path: ".env" });

async function main() {
  /*
  A ContractFactory in ethers.js is an abstraction used to deploy new smart contracts,
  so verifyContract here is a factory for instances of our Verify contract.
  */
  const verifyContract = await ethers.getContractFactory("Verify");

  // deploy the contract
  const deployedVerifyContract = await verifyContract.deploy();

  await deployedVerifyContract.deployed();

  // print the address of the deployed contract
  console.log("Verify Contract Address:", deployedVerifyContract.address);

  console.log("Sleeping.....");
  // Wait for etherscan to notice that the contract has been deployed
  await sleep(10000);

  // Verify the contract after deploying
  await hre.run("verify:verify", {
    address: deployedVerifyContract.address,
    constructorArguments: [],
  });
}

function sleep(ms) {
  return new Promise((resolve) => setTimeout(resolve, ms));
}

// Call the main function and catch if there is any error
main()
  .then(() => process.exit(0))
  .catch((error) => {
    console.error(error);
    process.exit(1);
  });
```

Now open the `hardhat.config.js` file, we will add the `mumbai` network here so that we can deploy our contract to mumbai and an `etherscan` object so that we can verify our contract on `polygonscan`. Replace all the lines in the `hardhat.config.js` file with the given below lines.

```
require("@nomicfoundation/hardhat-toolbox");
require("dotenv").config({ path: ".env" });

const QUICKNODE_HTTP_URL = process.env.QUICKNODE_HTTP_URL;
const PRIVATE_KEY = process.env.PRIVATE_KEY;
const POLYGONSCAN_KEY = process.env.POLYGONSCAN_KEY;

module.exports = {
  solidity: "0.8.4",
  networks: {
    mumbai: {
      url: QUICKNODE_HTTP_URL,
      accounts: [PRIVATE_KEY],
    },
  },
  etherscan: {
    apiKey: {
      polygonMumbai: POLYGONSCAN_KEY,
    },
  },
};
```

Compile the contract, open up a terminal pointing at `etherscan-verification` directory and execute this command

```
npx hardhat compile
```

To deploy, open up a terminal pointing at `etherscan-verification` directory and execute this command

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

It should have printed a link to Mumbai PolygonScan, your contract is now verified. Click on the link and interact with your contract there 🚀🚀🚀

To pass the skill test for this level, select the `Mumbai` test network below and input the contract address you have verified on the Polygon Mumbai Testnet.

Share the link in Discord, and as always, feel free to ask any questions!

Submit Practical

Verify your smart contract address to pass the assessment for this level.

Ethereum Rinkeby | 0x... | Submit