

Ethereum Basics: Setting Up a Development Environment

1 Setup

In this exercise sheet, you are asked to set up an environment for developing smart contracts on the Ethereum blockchain. While we advise you to use the tools we propose, you are free to use any tool you want to.

Note: Be careful about the format when copying code or commands from the exercise sheet.

Recommended Software:

- 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 - <https://www.npmjs.com/>
Node Package Manager (NPM) comes installed with Node.js.
- Truffle - <https://trufflesuite.com/docs/truffle/quickstart/>
Truffle is a smart contract development framework for Ethereum, aiming to make life as an Ethereum developer easier.
- Ganache by Truffle - <https://trufflesuite.com/ganache/>
A personal blockchain for Ethereum development that you can use to deploy contracts, develop applications, and run tests. It is available as both a desktop application as well as a command-line tool.
- MetaMask - <https://metamask.io/>
MetaMask is an extension for accessing Ethereum-enabled distributed applications (dApps) in your browser. The extension injects the Ethereum web3 API into every website's JavaScript context, so that dApps can read from the blockchain. MetaMask also lets the user create and manage their own identities, so when a dApp wants to perform a transaction and write to the blockchain, the user gets a secure interface to review the transaction, before approving or rejecting it.
- Visual Studio Code - <https://code.visualstudio.com/>
- Visual Studio Code Extension: *Solidity*

Recommended Services:

- Infura - <https://infura.io/>
Provides instant access over HTTPS and WebSockets to the Ethereum network. Alternative to running your own full node.

Task Deploy a contract to the Sepolia test network

Sepolia is the recommended default testnet for application development before deployment on Mainnet, the main Ethereum network. There are also other testnets available for Ethereum which have different properties than Sepolia.

- <https://sepolia.dev/>
- <https://sepolia.etherscan.io/>
- <https://ethereum.org/en/developers/docs/networks/>

2 Setting Up a Sepolia Account

- Install MetaMask.
- Configure your MetaMask account (Write down your seed phrase! You will need it later).
- Open MetaMask and switch the network to Sepolia.
- Find some Sepolia Test Ether using a Faucet. One option is using the Faucet provided by Infura (you need to open up an account first) at <https://www.infura.io/faucet/sepolia> or ask your fellow students for a small donation. For this exercise, 0.1 Sepolia Ether will be sufficient.
- Send 0.001 Sepolia Ether to 0xaCb576A3f9F9ffa1e2276A1C06C0Ade4fE2197f8 (our address on Sepolia) and inspect your transaction on Sepolia Etherscan.

3 Setting Up Your Local Development Environment

- Install Visual Studio Code.
- Install the Solidity extension inside Visual Studio Code.
- Install Truffle globally on your computer: `npm install -g truffle` (Check your Node.js and NPM version if you face a problem during installment)
- Create an empty working directory and initialize a bare Truffle project with no smart contracts in it.
 - Run `truffle init`
 - **Contracts:** Directory for Solidity smart contracts.
 - **Migrations:** Directory for scriptable deployment files.
 - **test:** Directory for test files for testing your application and contracts
 - **truffle-config.js:** Truffle configuration file
- Add the following Solidity smart contract (*Greeter.sol*) under your *contracts* directory.

```
//SPDX-License-Identifier: Unlicense
pragma solidity ^0.8.15;

contract Greeter {
    string private greeting;

    constructor (string memory _greeting) {
        greeting = _greeting;
    }

    function greet() public view returns (string memory) {
        return greeting;
    }
}
```

- Compile your smart contracts `truffle compile` (Make sure to use a 0.8 compiler version)
- You should now see your contract's ABIs under the *build* folder.
- Add the following script (*1_deploy.js*) under your *migrations* folder. This script tells Truffle how to deploy the *Greeter.sol* smart contract.

```
const Greeter = artifacts.require("Greeter");

module.exports = async function (deployer) {
  //deploy Greeter contract
  await deployer.deploy(Greeter, "Hello BBSE 2023!");
};
```

4 Deploying to the Truffle Built-in Blockchain

- Run `truffle develop`
- Check the console output. See all the available accounts and their associated private keys.
- Run `migrate` to deploy the Greeter contract.
- Check the console for the transaction receipt. Detect the deployer account address and the used gas amount.
- Exit with `ctrl+c`

5 Deploying to the Local Ganache Network

- Install Ganache <https://trufflesuite.com/ganache/> and run the desktop application.
- Quickstart a new blockchain and inspect all the available accounts.
- Run `truffle migrate` to deploy the Greeter contract to the Ganache network.
- Using the UI, find which address deployed the smart contract. Also, inspect the transaction that includes the creation of the Greeter contract.
- Check the ABI of the contract and find the *address* and the *transactionHash* info for the deployment to the Ganache network. Compare it with the data available on the Ganache UI.

6 Deploying to the Sepolia Test Network

- Open a free **Infura** account and create an Ethereum project. Find your Sepolia endpoint.
- Run `npm i dotenv`. We will store the environment variables using this library.
- Create a `.env` file under the root directory with following content.

```
PRIVATE_KEY_1 = <YOUR_PRIVATE_KEY>
INFURA_API_KEY = <YOUR_INFURA_API_KEY>
```

You can find your private key by clicking the Account details option on MetaMask.

- Edit `truffle-config.js` file to deploy to the remote networks like Sepolia. By default, Truffle is configured to deploy to the local Ganache network. In the provided configuration, we add Sepolia as another network that we can interact with.

```

require("dotenv").config();
const HDWalletProvider = require("@truffle/hdwallet-provider");

module.exports = {
  networks: {
    development: {
      host: "127.0.0.1", // Localhost (default: none)
      port: 7545, // Standard Ethereum port (default: none)
      network_id: "*", // Any network (default: none)
    },
    sepolia: {
      provider: function () {
        return new HDWalletProvider({
          privateKeys: [process.env.PRIVATE_KEY_1],
          providerOrUrl: 'https://sepolia.infura.io/v3/${process.env.INFURA_API_KEY}',
          numberOfAddresses: 1,
        });
      },
      gas: 5000000,
      gasPrice: 5000000000, // 5 gwei
      network_id: 11155111,
    },
  },

  // Configure your compilers
  compilers: {
    solc: {
      version: "0.8.15", // Fetch exact version from solc-bin (
        default: truffle's version)
    },
  },
};

```

- Run `npm i @truffle/hdwallet-provider` to install the truffle-hdwallet-provider. This library handles the account management process (e.g., configuring the account to sign the transactions to deploy the smart contracts) and network connection. When we deploy to Ganache, Truffle uses the first available account by default for issuing the deployment transactions.
- Run `truffle migrate --network sepolia` to deploy the Greeter contract to the Sepolia network.
- Once the deployment is done, check out MetaMask to see how your balance changed.
- Find the transaction hash on your console and search it on <https://sepolia.etherscan.io/>.
- Navigate to the contract page on Etherscan and find the bytecode. Check if it is the same with the bytecode in the ABI of the Greeter contract. There you will find a *bytecode* and a *deployedBytecode*. Try to understand what is the difference. (**Hint:** Inspect the contract deployment transaction!)