Environment Setup 🛠️

As a smart contract developer we need to compile, test, debug, and deploy our contracts. Then we need a front-end library to help us interact with our smart contract. There are many development frameworks and front-end libraries that help us with this, the most popular being truffle and web3.js.

While truffle and web3.js are used in many projects today, more and more developers are switching over to Hardhat and ethers.js for smart contract development so we will be using this stack moving forward. We still recommend playing around with truffle and web3.js. There are even SDKs out there that abstract away the complexity of developing contracts so you can write everything in just JavaScript, but it's important to learn the basics first.

If you have a Python background, check out Brownie.

We also need an end point to test our contracts and broadcast our contract interactions. We will be using POKT to broadcast our transactions to the ethereum rinkeby testnet since it is the closest decentralized solution out there and the leading node provider. Infura and Alchemy are centralized providers that broadcast your transactions through their own centralized API and hardware.

Note: This lesson will be lengthy so set aside 20-30 mins. Feel free to take breaks in between.

Setting Up Hardhat 👷

Hardhat is a suite of tools that allow us to build, debug, and deploy our smart contracts on the Ethereum blockchain. They have a huge plugin library and you can even `` `console.log()` `` right in your smart contract for debugging. Hardhat comes with its own local blockchain that you can test on, saving you a ton of time.

If you would like to get a head start you can use the official Hardhat documentation to install

Hardhat. Let's get started.

1. First make sure you have node version 12 or above and npm installed. Then install Hardhat.

```
node --version
npm --version
```

If you don't have node you can find the installaiton guides for Windows, macOS, and Linux here.

2. Make a directory for our project called bank-smartcontract and cd into it. Then initalize npm for our project and install Hardhat. We're installing dotenv to store our project's environment variables.

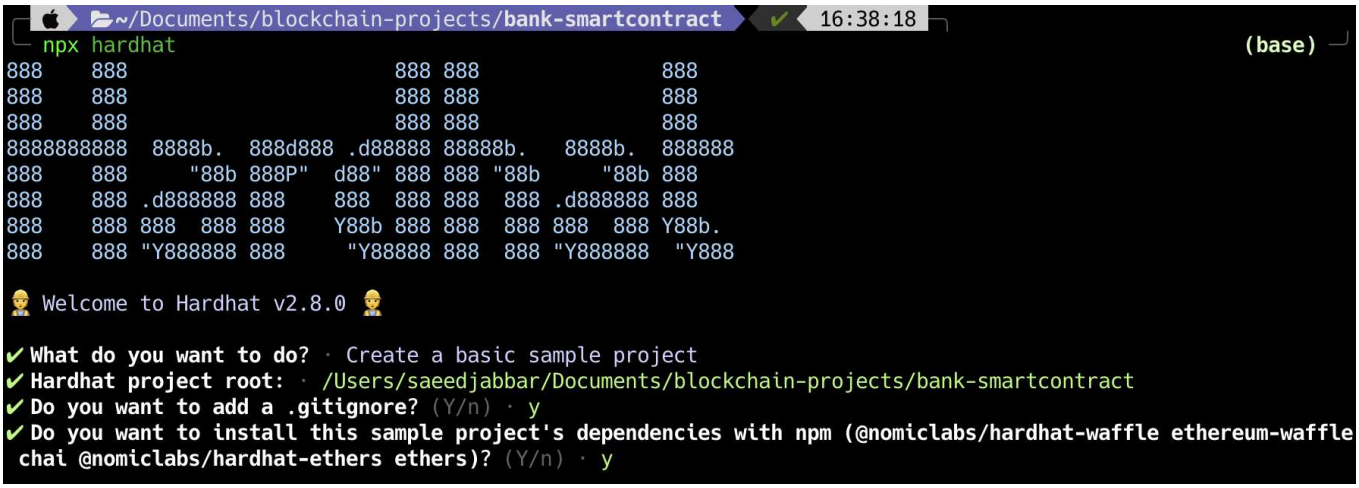
38%

```
cd bank-smartcontract
npm init -y
npm install dotenv
npm install --save-dev hardhat
```

3. Next run the following command to initialize a Hardhat project and then choose the first option `Create a sample project`

```
npx hardhat
```

Now hit enter for all the options and pay special attention to all the depencies being installed.



```
npm install --save-dev @nomiclabs/hardhat-waffle@^2.0.0 ethereum-waffle@^3.0.0
chai@^4.2.0 @nomiclabs/hardhat-ethers@^2.0.0 ethers@^5.0.0
```

We will do a quick breakdown of these dependencies and what they do.

Dependencies	What they do?
waffle	Used for testing smart contracts
chai	Used for testing in Javascript
ethers	A javascript library used to interact with the Ethereum blockchain.

4. Run `**npx hardhat**` again and you will see the global options and pay special attention to the tasks, we will using some of these often. you can use `**npx hardhat accounts**` to see a list of test accounts we can work with.

```
Usage: hardhat [GLOBAL OPTIONS] <TASK> [TASK OPTIONS]

GLOBAL OPTIONS:

  --config              A Hardhat config file.
  --emoji               Use emoji in messages.
  --help                Shows this message, or a task's help if its name is provided
  --max-memory          The maximum amount of memory that Hardhat can use.
  --network             The network to connect to.
  --show-stack-traces   Show stack traces.
  --tsconfig            A TypeScript config file.
  --verbose             Enables Hardhat verbose logging
  --version             Shows hardhat's version.


AVAILABLE TASKS:

  accounts      Prints the list of accounts
  check         Check whatever you need
  clean         Clears the cache and deletes all artifacts
  compile       Compiles the entire project, building all artifacts
  console       Opens a hardhat console
  flatten       Flattens and prints contracts and their dependencies
  help          Prints this message
  node          Starts a JSON-RPC server on top of Hardhat Network
  run           Runs a user-defined script after compiling the project
  test          Runs mocha tests

To get help for a specific task run: npx hardhat help [task]
```

5. Now open up your project folder in your favorite editor, we will be using VS Code for this class. Install this Solidity plugin for VS Code which allows us to format Solidity.

Let's check out the folder structure from Hardhat. `**contracts**` will be where our contracts go, `**scripts**` is where our scripts will live that allow us deploy our contract, and `**test**` is for any testing we would like to do.

6. Delete `Greeter.sol` from the contracts folder, delete `sample-script.js` from scripts, and `sample-test.js` from the test folder.

We're almost at the finish line. We're now going to setup our Hardhat config and our environment variables.

7. Go into hardhat.config.js delete everything and paste the following in and save:

```
require('@nomiclabs/hardhat-waffle');
require('dotenv').config();

module.exports = {
  solidity: "0.8.0",
  networks: {
    rinkeby: {
      url: `${process.env.POKT_RINKEBY_URL}`,
      accounts: [`${process.env.RINKEBY_PRIVATE_KEY}`],
    }
  }
};
```

This is straightforward, we're using solidity 8.0, the rinkeby test network, the url property is an Ethereum node which we will be using Alchemy for (more on this later) and in accounts will be the private key of an Ethereum account (more on this later).

We can access these enviroment globally using the following line of code:

```
require('dotenv').config();
```

🚨 SECURITY ALERT 🚨

Before we proceed, create a MetaMask account that will only be used for this course as your private key will be needed to sign transactions. Anyone who has your private key can steal all funds from your wallet, hence why we're using dotenv to store our private keys.

8. Now we're going to setup our enviromment variables. Create a file called `.env` in the root of our project folder and paste the following in:

```
POKT_RINKEBY_URL=YOUR_POKT_RINKEBY_URL
```

```
1 POKT_RINKEBY_URL=YOUR_POKT_RINKEBY_URL
2 RINKEBY_PRIVATE_KEY=YOUR_PRIVATE_KEY
```

Switch your MetaMask to the Rinkeby Testnet and paste in your private key. Here's how you can get your private key. Again make this a throwaway account that will never be used for real transactions. We will need the private key to sign our transactions, deploy and make function calls to our contract.

Setting up POKT 🦹‍♂️

Next we need an Ethereum node to broadcast our contract transaction to the blockchain so miners can pick it up, mine it, and store it on the blockchain. For this we will be using POKT. Head over there and sign up for an account and verify your email. Below is a video walk through of how to set up your end point with POKT.

1. Create an app and choose Ethereum Rinkeby for your chain, give your app a name and click launch app. Give it some time to bootup.
2. Click app security, then in approved chains make sure to select Ethereum Rinkeby and hit save changes.
3. Copy and paste your HTTP end point key into the `POKT_RINKEBY_URL` section. Save your .env file.

Here are some faucets from our previous lessons to load up your account with fake Eth or ask us in the discord.

Great job getting this far! That was a lot, so feel free to take a breather

Previous Next Mark Lesson As Completed