Smart Contract Developer Bootcamp Weekend Track: Day 2 Exercises: Hardhat

Software Installation **2**

Exercise 1: My First Hardhat Project **2**
- Setting Up Hardhat 2
- Creating The Smart Contract 8
- Deploying and Interacting With the Smart Contract 9
- Bonus Exercises: 12

Exercise 2: Hardhat Starter Kit **14**
- Downloading the Hardhat Starter Kit 14
- Setting Up The Hardhat Starter Kit 16
- Deploying The Smart Contracts 19
- Interacting With The Deployed Smart Contracts 19
  - Price Feed Consumer Contract 19
  - API Consumer Contract 20
  - Random Number Consumer Contract 22
- Bonus Exercises: 23

Exercise 3: Deploying to a Local Network **24**
- Setting up the Local Hardhat Network 24
- Deploying and Interacting With the Smart Contracts 27
  - Price Feed Consumer Contract 28
- Forking Ethereum Mainnet 28
- Bonus Exercises: 33

Exercise 4: Testing Smart Contracts **34**
- Installing Yarn 34
- Executing the Unit Tests 35
- Executing the Integration Tests 35
- Checking the Solidity Coverage 37
- Checking in your Project To a Public Code Repository 39
- Bonus Exercise: 46

Appendix: Troubleshooting **53**
- Deploying compiled contracts 53
- Starting the Local Hardhat Network 54
- Installing Yarn 55

Software Installation

Please ensure you've completed the setup steps before proceeding.

Exercise 1: My First Hardhat Project

In this exercise, you'll use Hardhat to create a new project that contains a simple smart contract that stores and retrieves a value, then you'll deploy it to the Kovan network and interact with it.

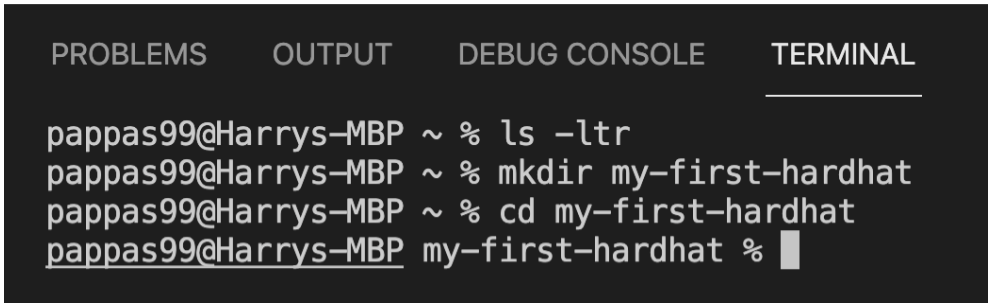
Setting Up Hardhat

1. Open Visual Studio Code
2. On the top header menu, choose View -> Terminal to bring up the VS Terminal (or press CTRL + `)
3. In your terminal, create a new directory called 'my-first-hardhat' using the mkdir command.

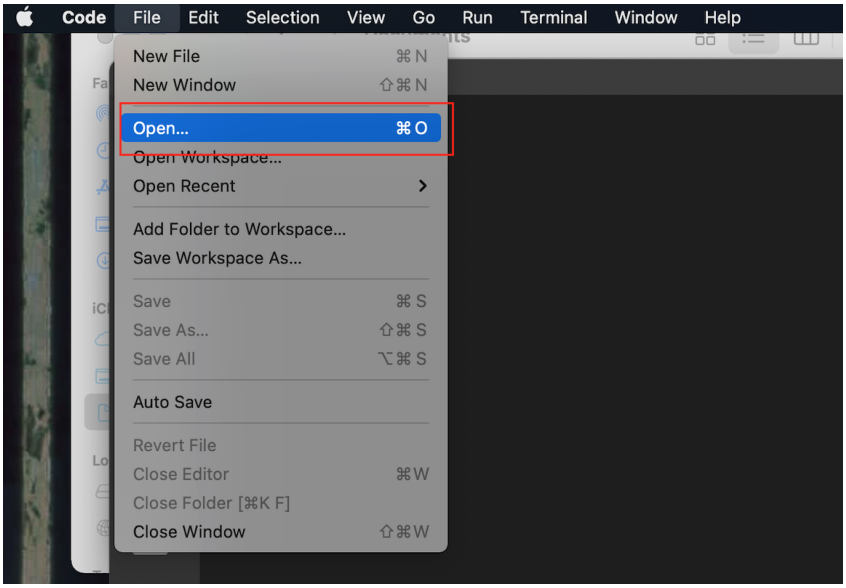
```
mkdir my-first-hardhat
```

4. Head into the directory by typing 'cd my-first-hardhat'

```
cd my-first-hardhat
```



5. In the top VS menu, choose File->Open (or Open Folder), then find your 'my-first-hardhat' directory and choose Open. You should now see an explorer menu on the left hand side. Accept any pop-up windows asking you if you trust the content in the directory. If the terminal in VS Code is gone, re-open it by going to View->Terminal



6. We're now ready to initiate a new Hardhat project. Type the following commands into the terminal to download the hardhat libraries to the current directory, and initiate a new hardhat project. The second one may take a while to complete. You can ignore any warnings about vulnerabilities.

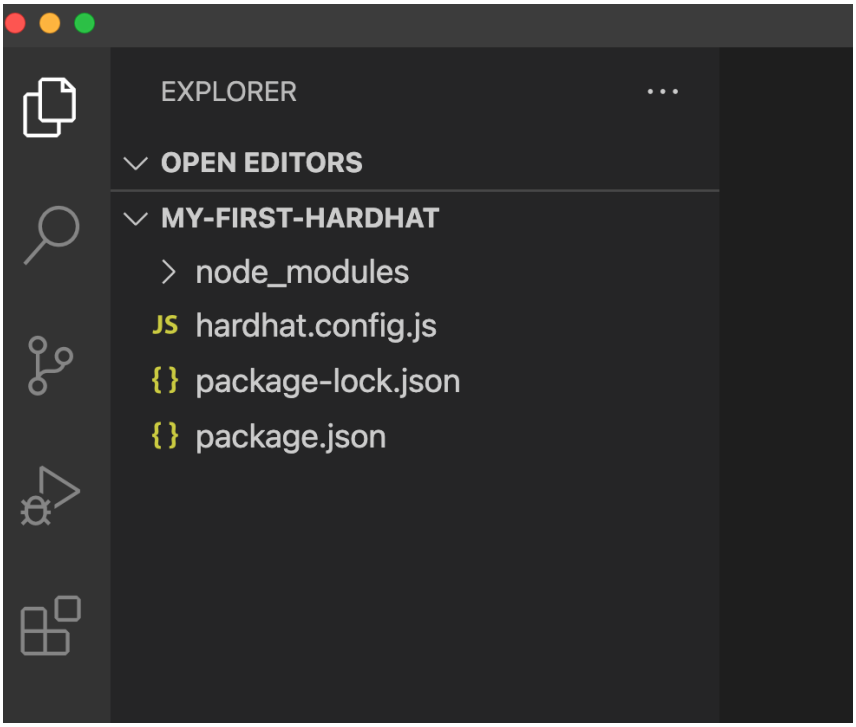
```
npm init --yes
```

```
npm install --save-dev hardhat
```

```
npx hardhat
```

7. Using the arrow keys, select the 'Create an empty hardhat.config.js' option and hit enter

8. You should now see the following folder and file structure in your explorer



9. For this exercise we're going to use the Ethers.js plugin. This will allow us to interact with Ethereum from outside the blockchain, using JavaScript. Run the following command in your terminal to install the ethers.js library, you can ignore any listed vulnerabilities.

```
npm install --save-dev @nomiclabs/hardhat-ethers ethers
```

10. Another library we'll need is the 'dotenv' library, which facilitates the loading of sensitive configuration data (such as keys) from a config file separate to your code. Install the library using the following command

```
npm install --save-dev dotenv
```

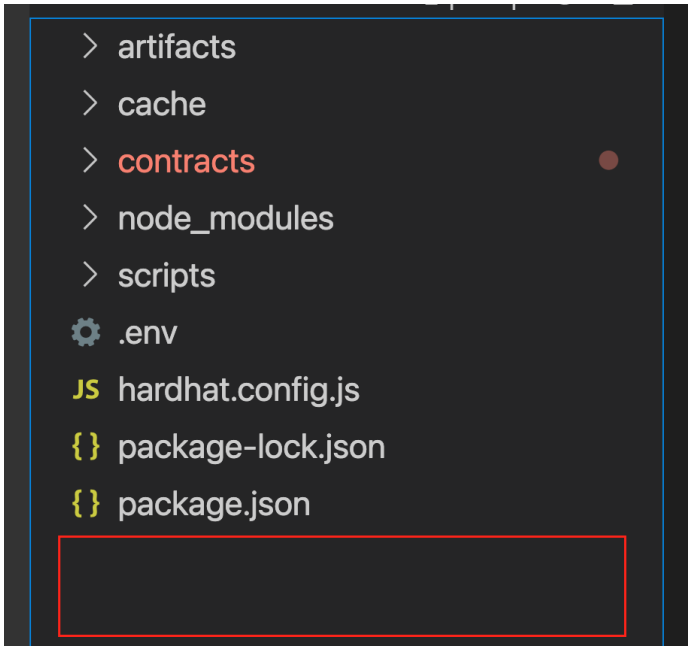
11. Open the **hardhat.config.js** file in the explorer. This file contains all the configuration for your hardhat project.

12. Replace the contents of the file with the following configuration, then save your changes. This configuration does the following:
    a. Tells hardhat to use the ethers library, as well as the dotenv library for loading environment variables from a .env config file

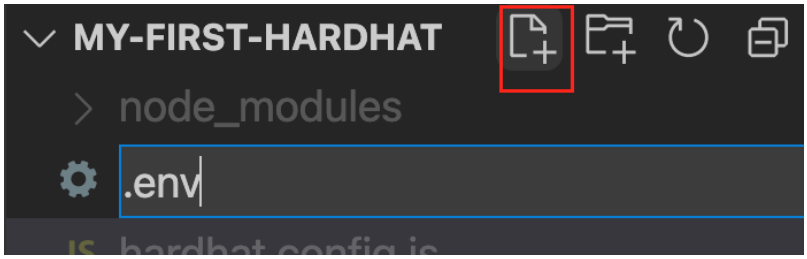
b. Specifies the networks that can be used for when we deploy smart contracts, in this case we will set the default to the Kovan testnet.
c. Specifies a solidity compiler version to use for compiling our smart contracts

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

/**
 * @type import('hardhat/config').HardhatUserConfig
 */
module.exports = {
  defaultNetwork: "kovan",
    networks: {
        hardhat: {
            // // If you want to do some forking, uncomment this
            // forking: {
            //   url: MAINNET_RPC_URL
            // }
        },
        localhost: {
        },
        kovan: {
            url: process.env.KOVAN_RPC_URL,
            accounts: [process.env.PRIVATE_KEY],
            saveDeployments: true,
        }
    },
  solidity: "0.7.3",
};
```

13. Now that we have a config file, we need to set up environment variables for our KOVAN_RPC_URL and PRIVATE_KEY environment variables listed in the config file. So that we don't include them in our main code, we're going to put them in a separate .env file that the config can read. This .env file will not be checked into any public code repositories. In the explorer, first click on the blank space under the 'MY-FIRST-HARDHAT' folder structure to ensure you've selected the top level folder to put your file in. You'll know the top level is selected if you get a blue outline over your entire folder structure



14. Then directly to the right of your folder name ‘MY-FIRST-HARDHAT’, press the new file button to create a new file, call it **.env**



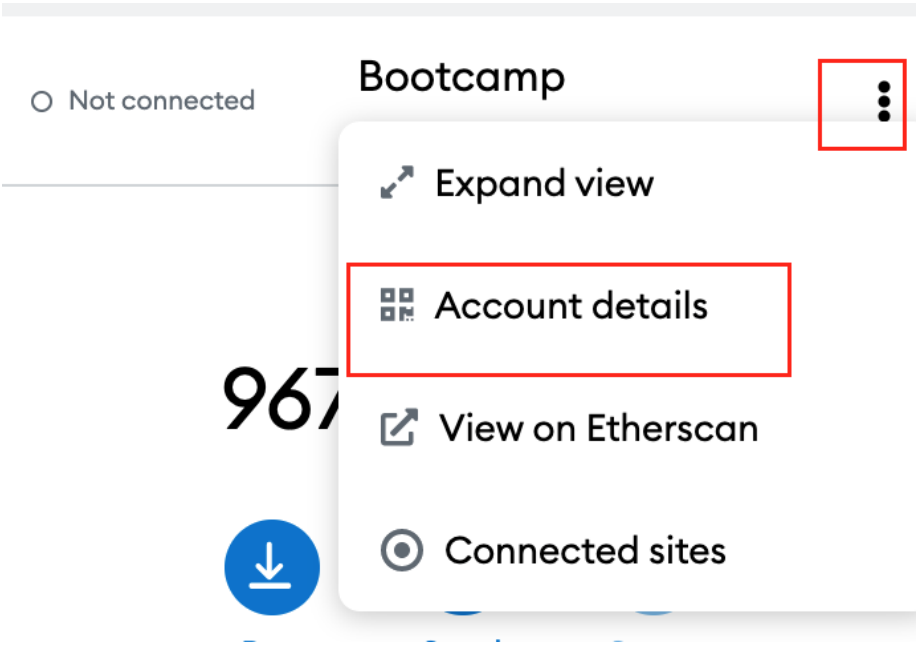
15. Open your newly created .env file in the editor by double clicking on it in the explorer window. Put the following text in the file

```
KOVAN_RPC_URL='abc'
PRIVATE_KEY='123'
```

16. Replace the contents of the KOVAN_RPC_URL string (abc), with the URL that you saved when you signed up for a free Infura key as part of the setup steps. It should be something like this: **https://kovan.infura.io/v3/soadhjfjdks8400975984slhfdskjdhf498**

17. Now we need to put our MetaMask wallet account private key in the PRIVATE_KEY environment variable. This will allow Hardhat to use our MetaMask wallet account to deploy and interact with contracts, so you don't have to manually use MetaMask and approve transactions. Open up MetaMask, press on the three dots to the right of your account name, then select Account Details -> Export private key, enter in your password, then copy your private key string, and paste it into your .env file PRIVATE_KEY string.

    NOTE: BE CAREFUL WITH COPYING AND PASTING PRIVATE KEYS FOR ACCOUNTS THAT HAVE MAINNET FUNDS IN THEM



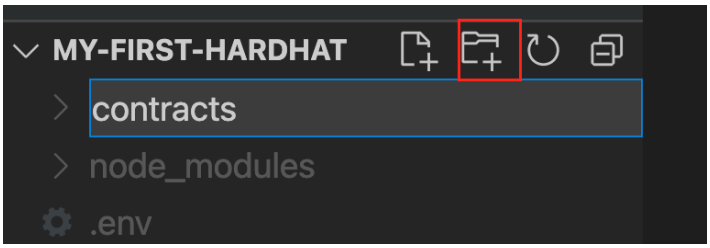
18. Your .env file should look something like this (but with different hash values). Save the file. Our hardhat configuration is now ready, and we're ready to create a smart contract

```
KOVAN_RPC_URL='https://kovan.infura.io/v3/334534klsdlkfhsd34043050u34slkdjf'
PRIVATE_KEY='43905lksjssfd403530LKHdslhs084k3l45hkl34h5kl34hlksmd48543085s'
```

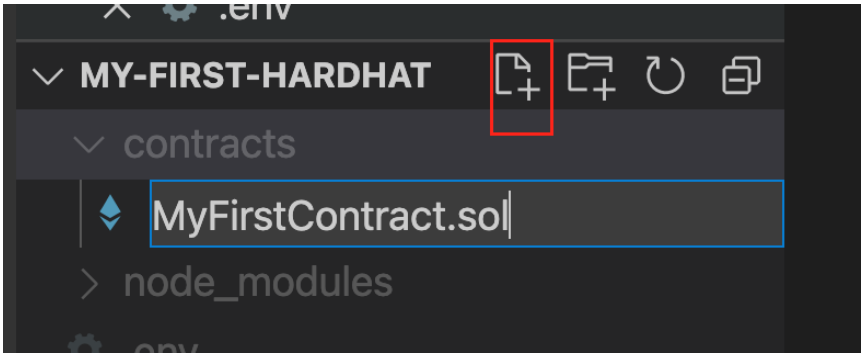
Creating The Smart Contract

1. In the explorer window on the left, click on the blank space under the ‘MY-FIRST-HARDHAT’ folder structure to ensure you’ve selected the top level folder. You’ll know the top level is selected if you get a blue outline over your entire folder structure

2. Then select the ‘New Folder’ icon, create a new folder called ‘contracts’. If you already have an empty contracts folder, you can skip this step.



3. Select the new contract folder in the explorer to ensure it’s highlighted, then select the ‘New File’ icon, and create a new file, call it ‘MyFirstContract.sol’, and press enter. Emsure it’s been created in the contracts folder.



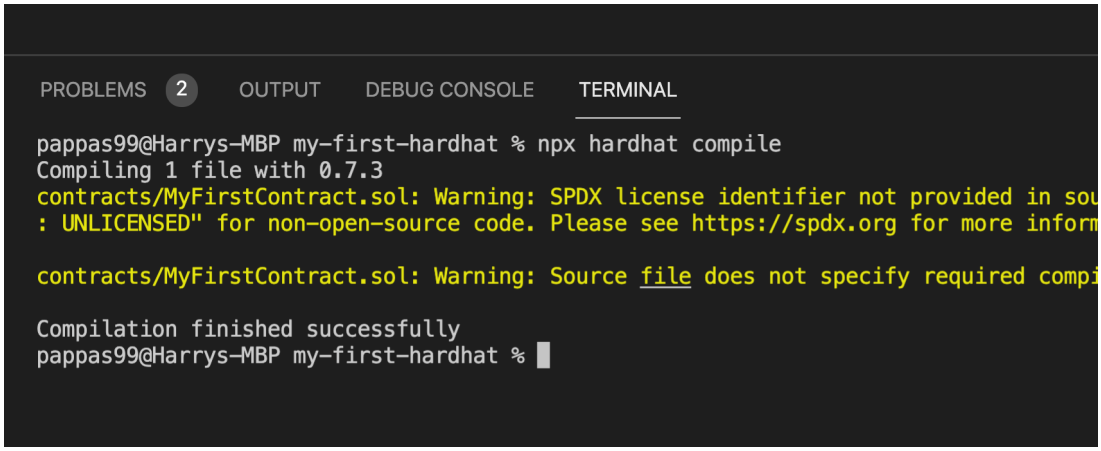
4. If it isn’t already open, open your new MyFirstContract.sol file by double clicking it in the explorer menu. Paste the following into the smart contract file, then save your changes. This is the same smart contract that you created in the first day of the bootcamp.

```
pragma solidity =0.7.3;
contract MyFirstContract {

    uint256 number;


    function setNumber(uint256 _num) public {
```

```
        number = _num;
    }


    function getNumber() public view returns (uint256){
        return number;
    }
}
```

5. Now that you've created your smart contract, you can compile it. In the terminal, type in the following command to compile your smart contract using the hardhat compiler.

```
npx hardhat compile
```

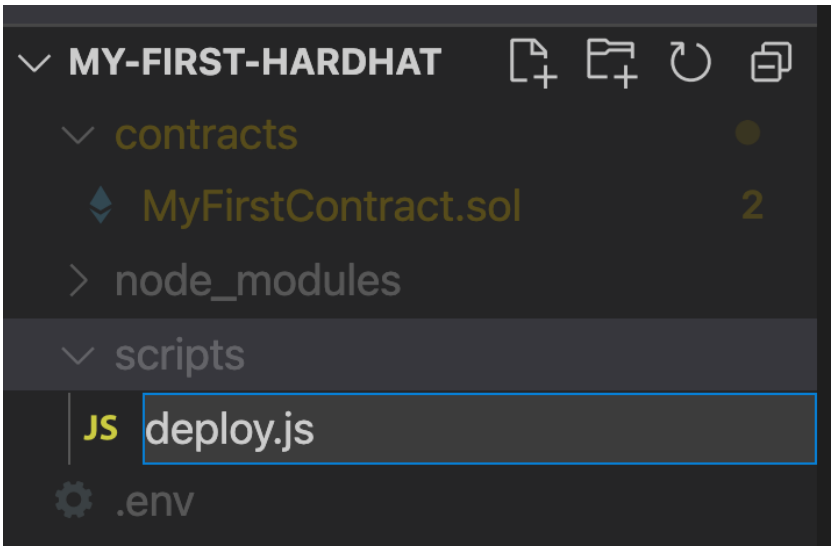


6. You now have our smart contract ready to be deployed! Next you'll create a script to deploy the smart contract to the Kovan network, and execute the two functions in it

Deploying and Interacting With the Smart Contract

Next you will create a script to deploy and then interact with your smart contract.

7. Similar to how you did previously, create a new folder in your project, call it 'scripts', ensuring it gets created at your topmost folder level in your project. Then inside the new folder, create a new file called 'deploy.js'.



8. Paste the following code into the newly created file, then save your changes. This script does the following:
    - Deploys your compiled smart contract
    - Obtains the deployed smart contract, and calls the ‘setNumber’ function
    - Calls the ‘getNumber’ function on the deployed contract

```
async function main() {

    const [deployer] = await ethers.getSigners();

    console.log(
      "Deploying contracts with the account:",
      deployer.address
    );

    console.log("Account balance:", (await
deployer.getBalance()).toString());

    const MyFirstContract = await
ethers.getContractFactory("MyFirstContract");
    const deployedContract = await MyFirstContract.deploy();
    console.log("Deployed MyFirstContract contract address:",
deployedContract.address);


    await deployedContract.setNumber(7)

    let result = BigInt(await
deployedContract.getNumber()).toString()
```

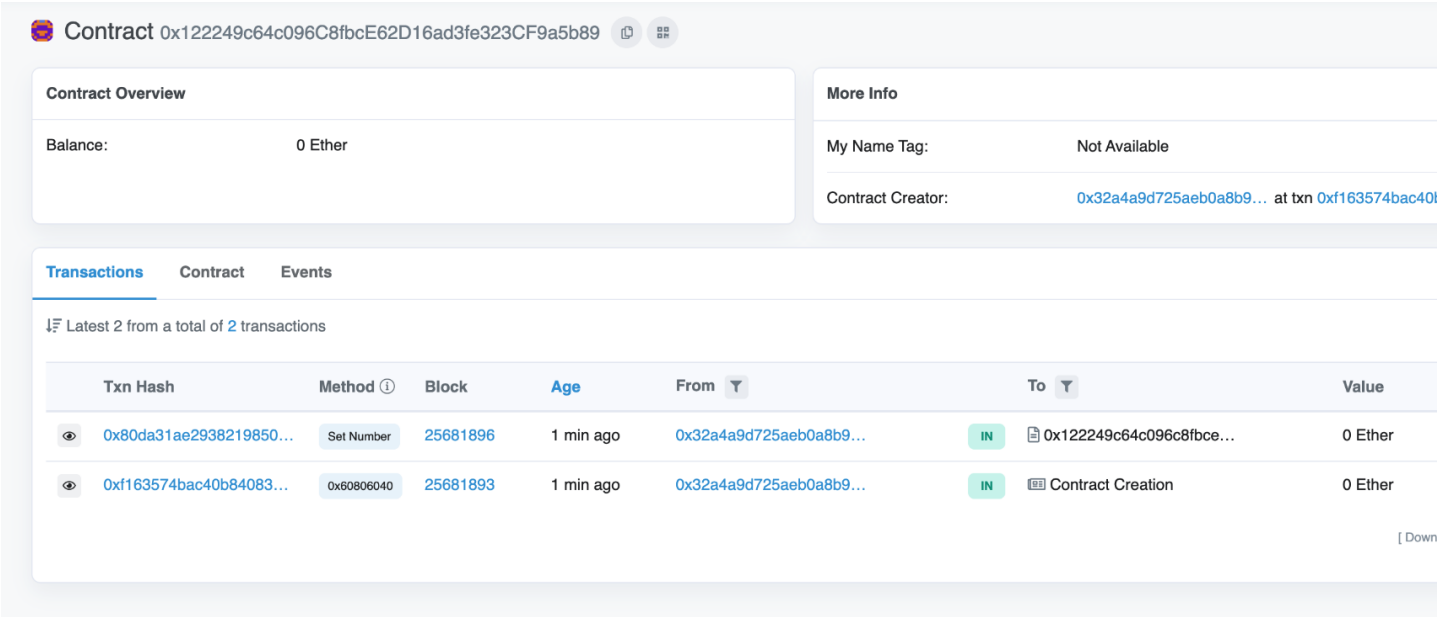
```
    console.log('Stored value in contract is: ', result)


}

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

9. You're now ready to deploy your smart contract to the Kovan network, and interact with it. Back in the terminal, enter the following command to deploy your smart contract to the Kovan network, and then execute the functions in the contract. If you don't specify a network in the command, Hardhat will use the default one in your hardhat.config.js file (which is set to the Kovan network). If you get an error stating you have insufficient funds, ensure your MetaMask account has some ETH in it, as per the setup instructions.

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

10. You should see your contract deployed to a new contract address on the Kovan network, and you should see your function calls successfully set and return a value in the smart contract.

11. Copy the deployed storage contract address specified in the output, and search for it on https://kovan.etherscan.io/. You should see two transactions, the creation of the contract, as well as a transaction to set the number listed in the transactions



12. Congratulations, you've successfully created a new smart contract project using the Hardhat Development Environment, deployed it to a public test network, and interacted with your deployed smart contract!

Bonus Exercises:

You can attempt to complete these exercises if you've completed the main exercise ahead of schedule:

1. Create a new contract in the contracts folder called ERC-20.sol. Open the file, and enter in the full source code for the ERC20 token contract example on the Ethereum developer tutorials page. The only changes you need to make is to update the pragma Solidity version at the top of the contract, and you should also comment out (put // at the beginning of the lines) the two 'Approval' and 'Transfer' events in the ERC20Basic contract to avoid possible compilation issues around duplicate events in the interface contract and the main contract. In addition to this, you can modify the token Symbol from ERC to whatever you choose. Save your file.

```
pragma solidity =0.7.3;

interface IERC20 {

    function totalSupply() external view returns (uint256);
    function balanceOf(address account) external view returns (uint256);
```

```
    function allowance(address owner, address spender) external view
returns (uint256);

    function transfer(address recipient, uint256 amount) external returns
(bool);
    function approve(address spender, uint256 amount) external returns
(bool);
    function transferFrom(address sender, address recipient, uint256
amount) external returns (bool);


    event Transfer(address indexed from, address indexed to, uint256
value);
    event Approval(address indexed owner, address indexed spender,
uint256 value);
}


contract ERC20Basic is IERC20 {

    string public constant name = "ERC20Basic";
    string public constant symbol = "ERC";
    uint8 public constant decimals = 18;


    //event Approval(address indexed tokenOwner, address indexed spender,
uint tokens);
    //event Transfer(address indexed from, address indexed to, uint
tokens);


    mapping(address => uint256) balances;

    mapping(address => mapping (address => uint256)) allowed;

    uint256 totalSupply_;

    using SafeMath for uint256;


   constructor(uint256 total) public {
```

```
    totalSupply_ = total;
    balances[msg.sender] = totalSupply_;
    }

    function totalSupply() public override view returns (uint256) {
    return totalSupply_;
    }

    function balanceOf(address tokenOwner) public override view returns
(uint256) {
        return balances[tokenOwner];
    }

    function transfer(address receiver, uint256 numTokens) public
override returns (bool) {
        require(numTokens <= balances[msg.sender]);
        balances[msg.sender] = balances[msg.sender].sub(numTokens);
        balances[receiver] = balances[receiver].add(numTokens);
        emit Transfer(msg.sender, receiver, numTokens);
        return true;
    }

    function approve(address delegate, uint256 numTokens) public override
returns (bool) {
        allowed[msg.sender][delegate] = numTokens;
        emit Approval(msg.sender, delegate, numTokens);
        return true;
    }

    function allowance(address owner, address delegate) public override
view returns (uint) {
        return allowed[owner][delegate];
    }

    function transferFrom(address owner, address buyer, uint256
numTokens) public override returns (bool) {
        require(numTokens <= balances[owner]);
        require(numTokens <= allowed[owner][msg.sender]);

        balances[owner] = balances[owner].sub(numTokens);
```

```
        allowed[owner][msg.sender] =
allowed[owner][msg.sender].sub(numTokens);
        balances[buyer] = balances[buyer].add(numTokens);
        emit Transfer(owner, buyer, numTokens);
        return true;
    }
}

library SafeMath {
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
      assert(b <= a);
      return a - b;
    }

    function add(uint256 a, uint256 b) internal pure returns (uint256) {
      uint256 c = a + b;
      assert(c >= a);
      return c;
    }
}
```

2. Create a new script in the scripts folder called ‘deploy-erc20.js’. Base the contents of the deploy script on your existing ‘deploy.js’ file.

3. Modify the deploy script so that it deploys your ‘ERC20Basic’ contract instead of the ‘MyFirstContract’ one. You should do the following:
    a. rename all variables specific to the storage contract to be specific to your new ERC20 one.
    b. Remove anything after the console.log statement that states which address the contract was deployed to
    c. Update the console.log statement to say ‘Deployed ERC20 contract address’
    d. In the .deploy command, pass in an initial total supply into the empty brackets, this will pass in the value to the constructor in the ERC20Basic contract

Spoiler/Solution (highlight to see)

4. Compile all contracts in your project, then execute your new script

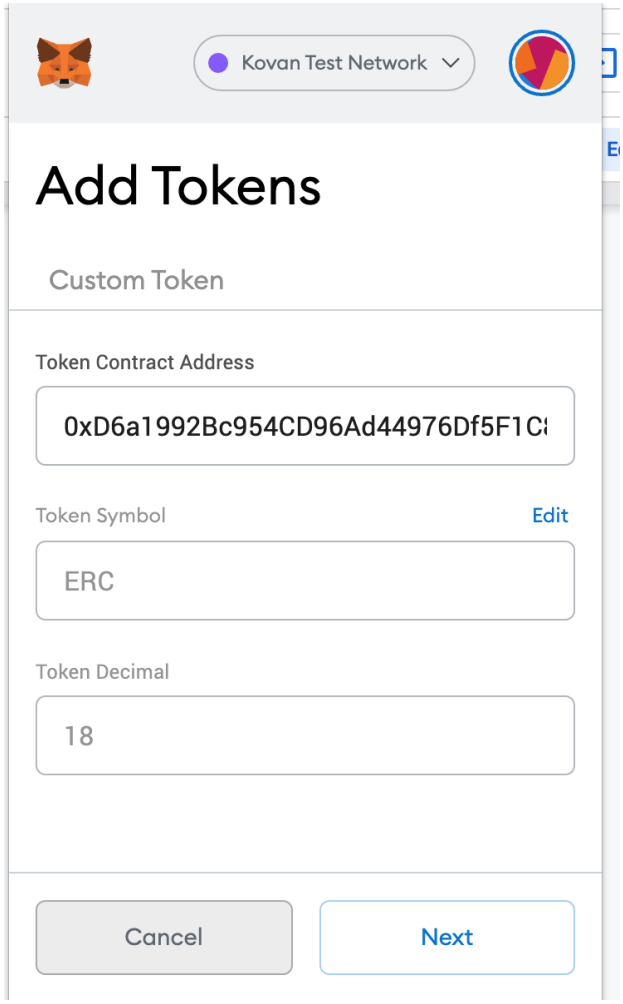
```
npx hardhat compile
```

```
npx hardhat run scripts/deploy-erc20.js
```

5. Look for your deployed contract on https://kovan.etherscan.io/.

6. Append to your new script to send yourself some tokens with the deployed contract by calling the required functions, and passing in the parameters. For now you can hardcode your wallet address values, you can obtain them from MetaMask. I.e send tokens from your current account that you have setup in the private key environment variable, and send them to another wallet address in your MetaMask. Check your transaction in Etherscan.
7. Copy the deployed address of your token, and add the token to your MetaMask wallet with the ‘Add Token’ button at the bottom of the Assets section. Paste in your contract

address and add the token. You should see the balance of the token in your assets section



Exercise 2: Hardhat Starter Kit

In this exercise, you'll download the Hardhat Starter Kit, and use it to create, deploy and execute smart contracts that use Chainlink Data Feeds, Any-API and VRF. This will give you some real world experience in using hardhat to deploy and interact with smart contracts using hardhats built in tasks feature.

Downloading the Hardhat Starter Kit

1. In VS Code, go to your terminal, then go back to your home folder by typing in 'cd ..' and pressing enter

```
cd ..
```

2. Pull a copy of the Hardhat Starter Kit with the following command:

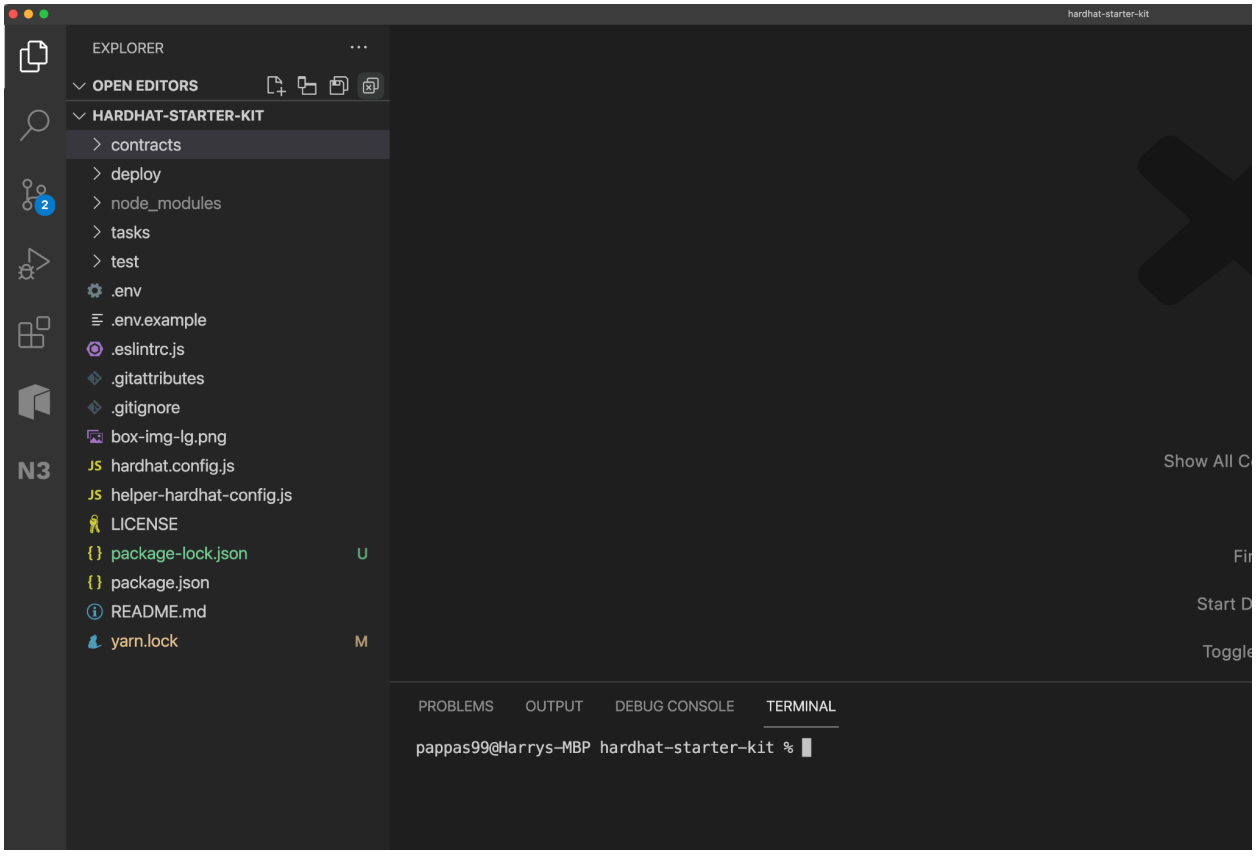
```
git clone
https://github.com/smartcontractkit/hardhat-starter-kit
```

```
pappas99@Harrys-MBP ~ % git clone https://github.com/smartcontractkit/hardhat-starter-kit
Cloning into 'hardhat-starter-kit'...
remote: Enumerating objects: 30058, done.
remote: Counting objects: 100% (30058/30058), done.
remote: Compressing objects: 100% (20856/20856), done.
remote: Total 30058 (delta 7687), reused 30022 (delta 7655), pack-reused 0
Receiving objects: 100% (30058/30058), 64.23 MiB | 1.07 MiB/s, done.
Resolving deltas: 100% (7687/7687), done.
pappas99@Harrys-MBP ~ %
```

3. In your terminal, you need to change the directory to your copy of the Hardhat Starter Kit that you downloaded in the previous step. Then you need to install all of the library dependencies that the starter kit requires. Run the following commands. The second one that installs the libraries may take a couple of minutes. You can ignore any listed vulnerabilities.

```
cd hardhat-starter-kit
npm install
```

4. Because we now have a new project in a new folder, we need to switch to it in Visual Studio Code. Choose File-> Open, find the ‘hardhat-starter-kit- folder, select it and press Open. If your VS Terminal has disappeared, re-open it by selecting View-> Terminal (or pressing CTRL + `). Your VS Code should look like this:



Setting Up The Hardhat Starter Kit

1. Open the **hardhat.config.js** file in the explorer. This file contains all the configuration for your hardhat project.

2. In the networks section where the Kovan network is listed, you should comment out the lines that mention MNEMONIC, and uncomment out the line that mentions PRIVATE_KEY, then save the file. This is because you're going to use your PRIVATE_KEY value from our MetaMask account to deploy smart contracts. In addition to this, you should ensure the ‘defaultNetwork’ field is set to ‘Kovan’, as we will be deploying to the Kovan network. The config file should look exactly like this, changes are in bold

```
/**
 * @type import('hardhat/config').HardhatUserConfig
 */
require("@nomiclabs/hardhat-waffle")
```

```
require("@nomiclabs/hardhat-ethers")
require("@nomiclabs/hardhat-truffle5")
require("@nomiclabs/hardhat-etherscan")
require("hardhat-deploy")
require("./tasks/accounts")
require("./tasks/balance")
require("./tasks/fund-link")
require("./tasks/withdraw-link")
require("./tasks/block-number")
require("./tasks/block-number")
require("./tasks/random-number-consumer")
require("./tasks/price-consumer")
require("./tasks/api-consumer")


require('dotenv').config()

const MAINNET_RPC_URL = process.env.MAINNET_RPC_URL ||
process.env.ALCHEMY_MAINNET_RPC_URL ||
"https://eth-mainnet.alchemyapi.io/v2/your-api-key"
const RINKEBY_RPC_URL = process.env.RINKEBY_RPC_URL ||
"https://eth-rinkeby.alchemyapi.io/v2/your-api-key"
const KOVAN_RPC_URL = process.env.KOVAN_RPC_URL ||
"https://eth-kovan.alchemyapi.io/v2/your-api-key"
const MNEMONIC = process.env.MNEMONIC || "your mnemonic"
const ETHERSCAN_API_KEY = process.env.ETHERSCAN_API_KEY || "Your
etherscan API key"
// optional
const PRIVATE_KEY = process.env.PRIVATE_KEY || "your private key"

module.exports = {
    defaultNetwork: "kovan",
    networks: {
        hardhat: {
            // // If you want to do some forking, uncomment this
            // forking: {
            //   url: MAINNET_RPC_URL
            // }
        },
        localhost: {
        },
```

```
        kovan: {
            url: KOVAN_RPC_URL,
             accounts: [PRIVATE_KEY],
            //accounts: {
            //    mnemonic: MNEMONIC,
            //},
            saveDeployments: true,
        },
        rinkeby: {
            url: RINKEBY_RPC_URL,
            // accounts: [PRIVATE_KEY],
            accounts: {
                mnemonic: MNEMONIC,
            },
            saveDeployments: true,
        },
        ganache: {
            url: 'http://localhost:8545',
            accounts: {
                mnemonic: MNEMONIC,
            }
        }
    },
    etherscan: {
        // Your API key for Etherscan
        // Obtain one at https://etherscan.io/
        apiKey: ETHERSCAN_API_KEY
    },
    namedAccounts: {
        deployer: {
            default: 0, // here this will by default take the first
account as deployer
            1: 0 // similarly on mainnet it will take the first account
as deployer. Note though that depending on how hardhat network are
configured, the account 0 on one network can be different than on another
        },
        feeCollector: {
            default: 1
        }
    },
    solidity: {
```

```
        compilers: [
            {
                version: "0.6.6"
            },
            {
                version: "0.4.24"
            }
        ]
    },
    mocha: {
        timeout: 100000
    }
}
```

3. Next, you need to set up environment variables for your KOVAN_RPC_URL and PRIVATE_KEY environment variables listed in the config file. Similar to the last exercise, create a new file in the hardhat-starter-kit folder, call it **.env.** If a file already exists, simply open it and use that as your starting point.

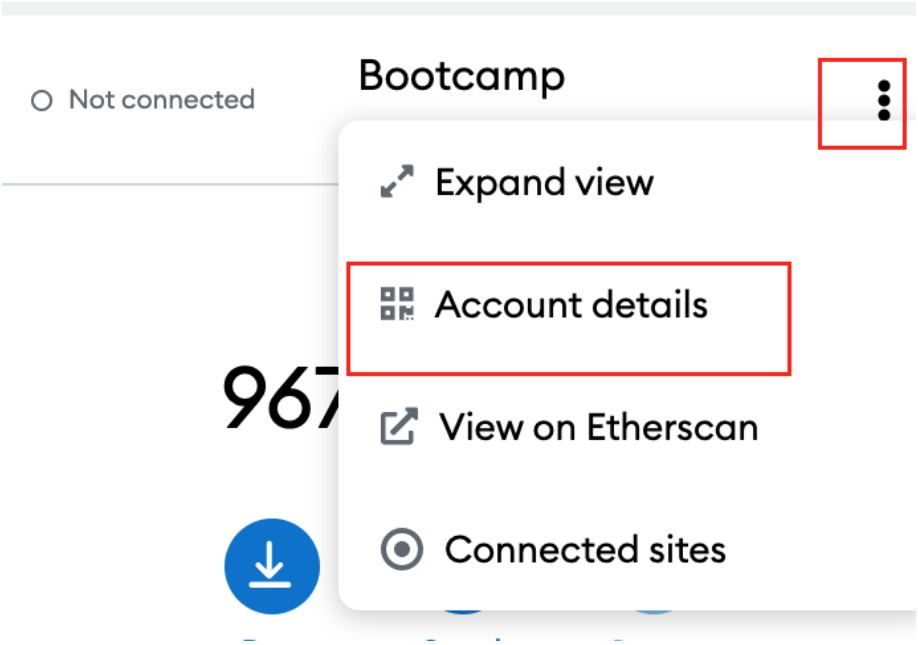
4. Open your .env file in the editor by double clicking on it in the explorer window. Put the following text in the file

```
KOVAN_RPC_URL='abc'
PRIVATE_KEY='123'
```

5. Replace the contents of the KOVAN_RPC_URL string (abc), with the URL that you saved when you signed up for a free Infura key as part of the setup steps. It should be something like this: **https://kovan.infura.io/v3/soadhjfjdks8400975984slhfdskjdhf498**

6. Now you need to put our MetaMask wallet account private key in the PRIVATE_KEY environment variable. Open up MetaMask, press on the three dots to the right of your account name, then select Account Details -> Export private key, enter in your password, then copy your private key string, and paste it into your .env file PRIVATE_KEY string.

   NOTE: BE CAREFUL WITH COPYING AND PASTING PRIVATE KEYS FOR ACCOUNTS THAT HAVE MAINNET FUNDS IN THEM



7. Your .env file should look something like this (but with different hash values). Save the file. Our hardhat starter kit configuration is now ready, and we're ready to deploy our smart contracts

```
KOVAN_RPC_URL='https://kovan.infura.io/v3/334534klsdlkfhsd34043050u34slkdjf'
PRIVATE_KEY='439051ksjssfd403530LKHdslhs084k3145hkl34h5kl34hlksmd48543085s'
```

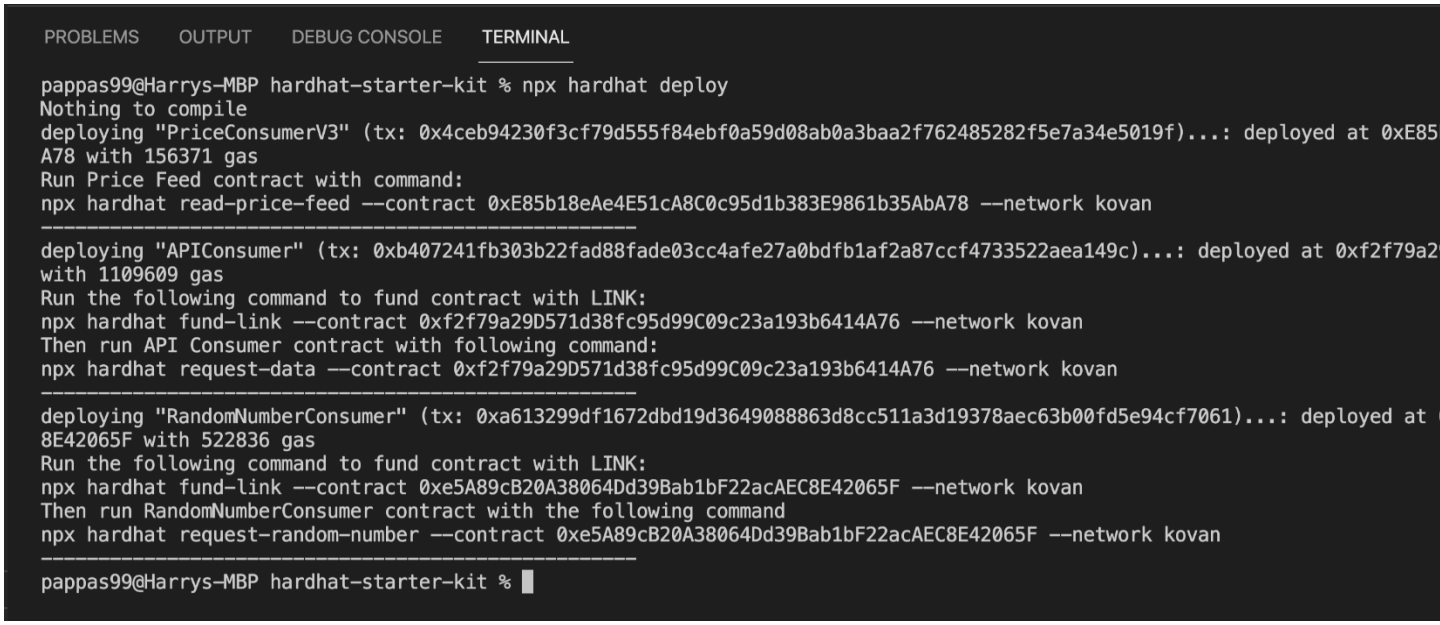
Deploying The Smart Contracts

1. Before you deploy the smart contracts in the hardhat starter kit to the Kovan test network, you should compile them first. Run the following command in the terminal

```
npx hardhat compile
```

2. Now that our contracts are compiled, run the following command to deploy the smart contracts in the hardhat starter kit to the Kovan test network. You should see output similar to the following screenshot

```
npx hardhat deploy
```



3. Your smart contracts that use Chainlink are now deployed to the Kovan network, and we're ready to start interacting with them!

Interacting With The Deployed Smart Contracts

Price Feed Consumer Contract

1. To interact with the deployed Price Feed Consumer contract, run the 'read-price-feed' task, which will query the deployed smart contract, and return the latest price of the specified price feed. You can do this by running the following command, replacing the contract address with the one specified in your console output for the deployed PriceConsumerV3 contract. Alternatively, the console output during deployment gives you the exact command to run to query the PriceConsumerV3 contract, you can copy and paste the command directly from there

```
npx hardhat read-price-feed --contract
replace-with-your-PriceConsumerV3-address --network kovan
```

```
pappas99@Harrys-MBP hardhat-starter-kit % npx hardhat read-price-feed --contract 0x
95d1b383E9861b35AbA78 --network kovan
Reading data from Price Feed consumer contract  0xE85b18eAe4E51cA8C0c95d1b383E9861b
  kovan
Price is:  193037238547
pappas99@Harrys-MBP hardhat-starter-kit %
```

API Consumer Contract

2. To interact with the deployed API Consumer contract, first run the ‘request-data’ task to execute the getVolumeData function, which will reach out to the crypto-compare API to obtain the current ETH/USD volume. You can do this by running the following command, replacing the contract address with the one specified in your console output for the deployed APIConsumer contract. Alternatively, the console output during deployment gives you the exact command to run to interact with the APIConsumer contract, you can copy and paste the command directly from there

```
npx hardhat request-data --contract replace-with-your-APIConsumer-address
--network kovan
```

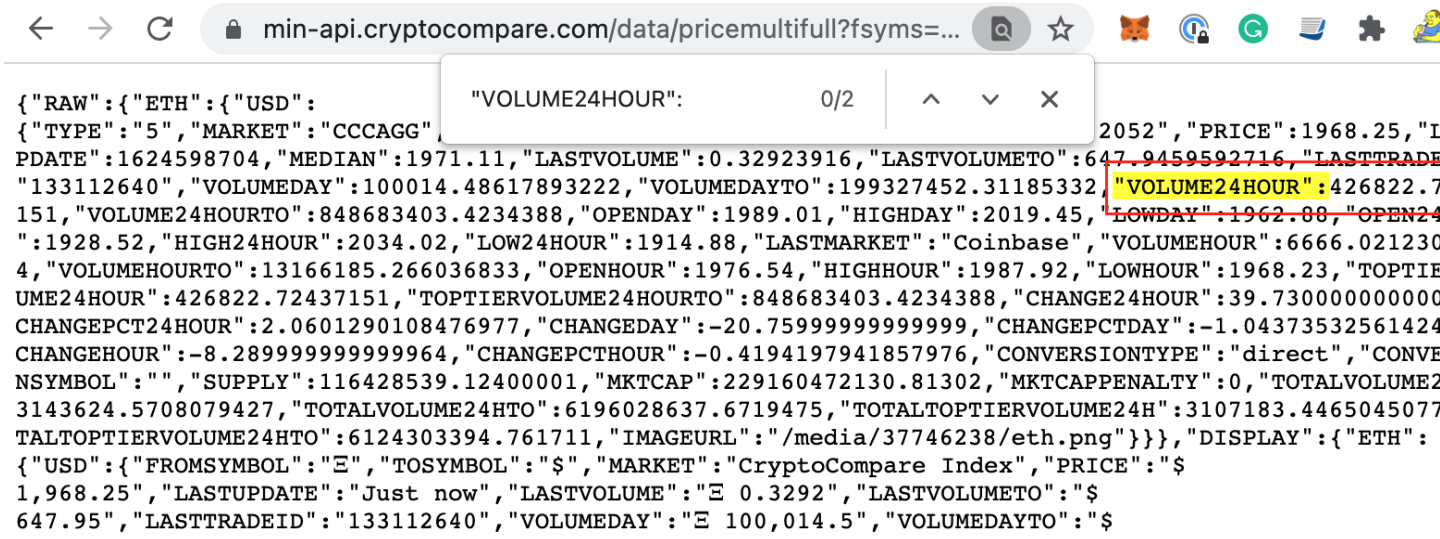
```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

Calling API Consumer contract  0xc036A5A16F0661fB8eE3e2bC6Ee37483bC3530d1  on n
Contract  0xc036A5A16F0661fB8eE3e2bC6Ee37483bC3530d1  external data request suc
742812c08e55f0edfa6261028feab956ab5b88244b8adc451bd3b6
Run the following to read the returned result:
npx hardhat read-data --contract 0xc036A5A16F0661fB8eE3e2bC6Ee37483bC3530d1 --n
pappas99@Harrys-MBP chainlink-hardhat-box-1 %
```

3. Now that you’ve performed an external data request, you can execute the ‘read-data’ task to call the volume function in the API Consumer smart contract, and see the result of the external data request. If you get a 0 result, wait 30 seconds and try again, sometimes the callback from the Oracle to Ethereum can take a few seconds for the transaction to confirm.

```
npx hardhat read-data --contract replace-with-your-APIConsumer-address
--network kovan
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

pappas99@Harrys-MBP chainlink-hardhat-box-1 % npx hardhat read-data --contract 0xc036A5A16F066
ork kovan
Reading data from API Consumer contract  0xc036A5A16F0661fB8eE3e2bC6Ee37483bC3530d1  on netwo
Data is:  425904519738440000000000
pappas99@Harrys-MBP chainlink-hardhat-box-1 %
```

4. You have now successfully performed a request for external data, and obtained the result into your smart contract using a Chainlink oracle. You can verify the result by opening a browser window, heading to https://min-api.cryptocompare.com/data/pricemultifull?fsyms=ETH&tsyms=USD and then searching for the string **"VOLUME24HOUR"**. As per the logic in the smart contract, the result has been multiplied by 10**18 to avoid any floating point numbers.

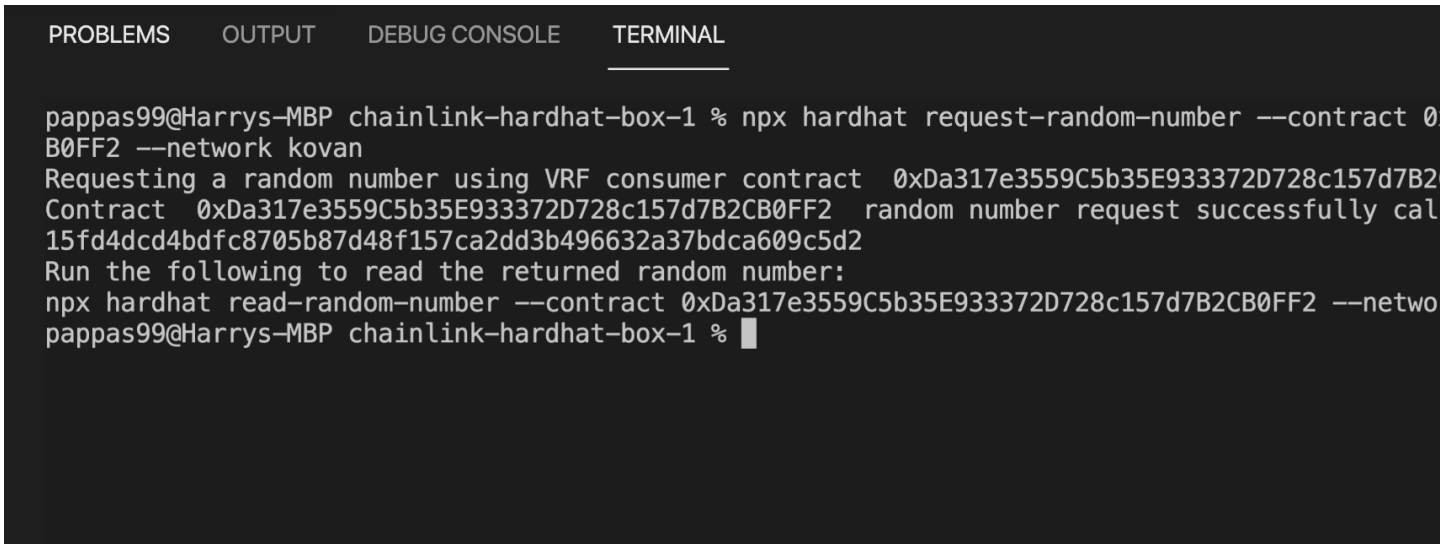


Random Number Consumer Contract

5. To interact with the deployed Random Number Consumer contract, first run the 'request-random-number' task to execute the getRandomNumber function, which will reach out to a Chainlink Oracle and make a request for randomness. You can do this by running the following command, replacing the contract address with the one specified in your console output for the deployed RandomNumberConsumer contract. Alternatively,

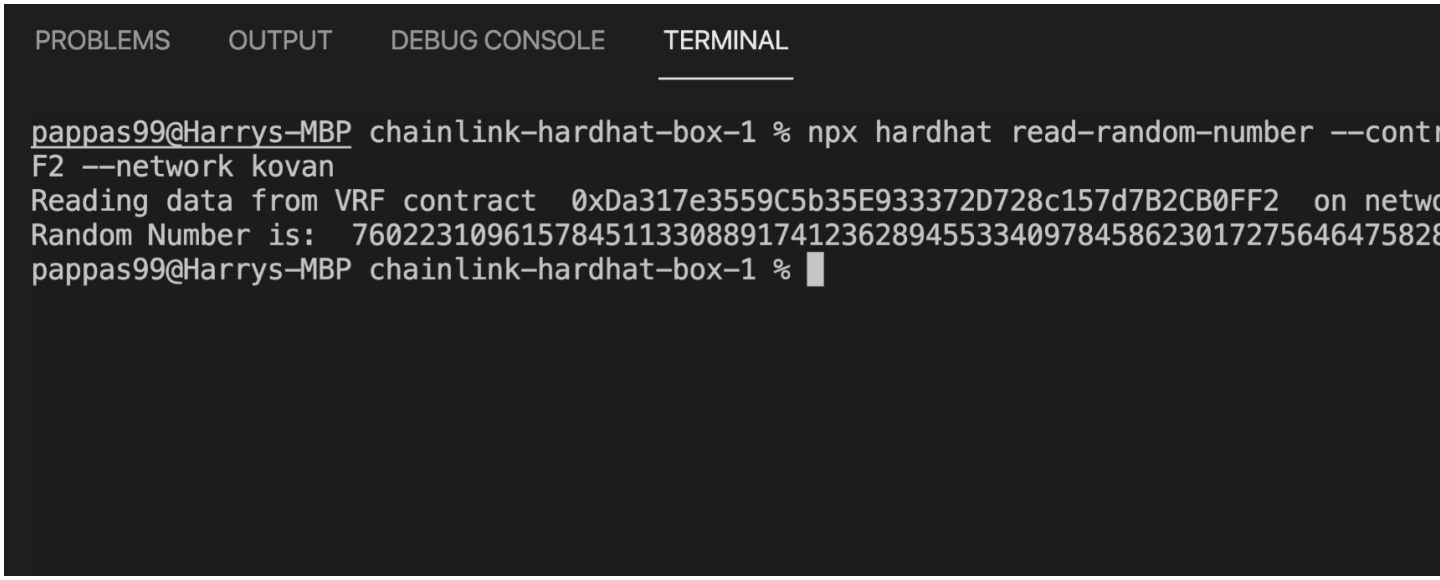
the console output during deployment gives you the exact command to run to interact with the RandomNumberConsumer contract, you can copy and paste the command directly from there

```
npx hardhat request-random-number --contract
replace-with-your-RandomNumberConsumer-address --network kovan
```



6. Now that you've performed a request for randomness, you can execute the 'read-random-number' task to call the randomResult function in the RandomNumberConsumer smart contract, and see the result of the random number request. If you get a 0 result, wait 30 seconds and try again, sometimes the callback from the Oracle to Ethereum can take a few seconds for the transaction to confirm.

```
npx hardhat read-random-number --contract
replace-with-your-RandomNumberConsumer-address --network kovan
```



Congratulations, you've successfully used the Hardhat Starter Kit to perform the following:

- Deploy three smart contracts to the Kovan testnet
- Interacted with a deployed smart contract to use Chainlink Data Feeds
- Interacted with a deployed smart contract to perform an API request using a Chainlink Oracle
- Interacted with a deployed smart contract to perform a request for randomness using Chainlink VRF

Bonus Exercises:

You can attempt to complete these exercises if you've completed the main exercise ahead of schedule:

1. Create a new contract in the contracts folder. You can create or put any contract inside it! If you want some examples, check out the Solidity docs examples.

2. Create a script in the deploy folder to deploy your contract

3. Compile and deploy your contracts to the kovan network. You may need to update your pragma statement in your new contract to match the version defined in the hardhat.config.js file (or update your hardhat.config.js solidity version to match your new contract pragma version).

```
npx hardhat compile
```

```
npx hardhat deploy
```

4. Create a new task for your contract in the tasks folder so you can interact with the contract once it's deployed

5. Add your task to the hardhat.config.js file at the top, where all the other tasks are added

```
require("./tasks/block-number")
require("./tasks/random-number-consumer")
require("./tasks/price-consumer")
require("./tasks/api-consumer")
...
```

6. Execute your task on the command line, passing in any defined parameters so you can interact with your deployed contract. Here is an example of executing a task called 'task-name' that takes a 'param1' parameter

```
npx hardhat task-name --param1 paramvalue --network kovan
```

Exercise 3: Deploying to a Local Network

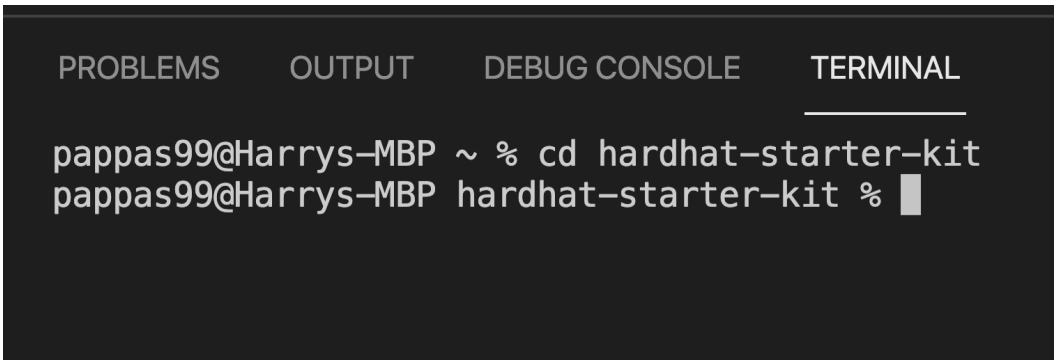
In this exercise, you'll get experience in starting a local hardhat network, deploying smart contracts to a local network, deploying smart contracts to a local network that's a fork of the mainnet, and interacting with smart contracts deployed locally.

Setting up the Local Hardhat Network

1. Open Visual Studio Code, ensure your current project is the 'hardhat-starter-kit' workspace from day 2 of the Developer Bootcamp (hint: If it's not, go File -> Open, and select the 'hardhat-starter-kit' folder)

2. If it's not already open, open the VS Terminal (View -> Terminal or CTRL + `)

3. If you're not already in it, head into your 'hardhat-starter-kit' folder in the terminal



4. Next, you need to ensure you've set the MAINNET_RPC_URL environment variable in the .env file. This will be used to create a fork of the Ethereum mainnet when we deploy to our local Hardhat network later on. Do this by taking the URL you noted from when you signed up to Alchemy in the [bootcamp setup instructions], and pasting it in an 'MAINNET_RPC_URL' variable in your .env file

```
MAINNET_RPC_URL='https://eth-mainnet.alchemyapi.io/v2/insert_key_here',
```

   Your .env file should have three lines in it

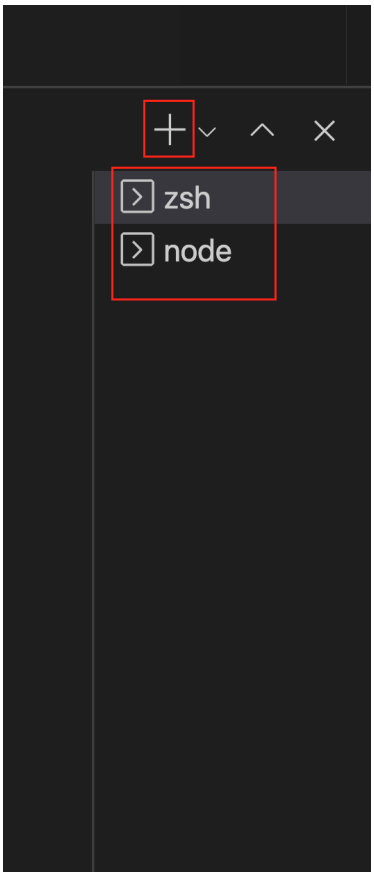
```
KOVAN_RPC_URL='https://kovan.infura.io/v3/insert_key_here
PRIVATE_KEY='private_key_hash_here'
MAINNET_RPC_URL='https://eth-mainnet.alchemyapi.io/v2/insert_key_here'
```

5. Before you start the local hardhat network, you need to modify the 'hardhat.config.js' file, and tell it to use the local Hardhat network by default. Open it up, and set the 'defaultNetwork' config to "hardhat". In addition to this, ensure you have the PRIVATE_KEY field uncommented out, and the MNEMONIC field commented out. It should match the code below (highlighted in bold):

```
module.exports = {
    defaultNetwork: "hardhat",
    networks: {
        hardhat: {
            // // If you want to do some forking, uncomment this
            // forking: {
            //   url: MAINNET_RPC_URL
            // }
        },
        localhost: {
```

```
    },
    kovan: {
        url: KOVAN_RPC_URL,
         accounts: [PRIVATE_KEY],
        //accounts: {
        //    mnemonic: MNEMONIC,
        //},
        saveDeployments: true,
    },
```

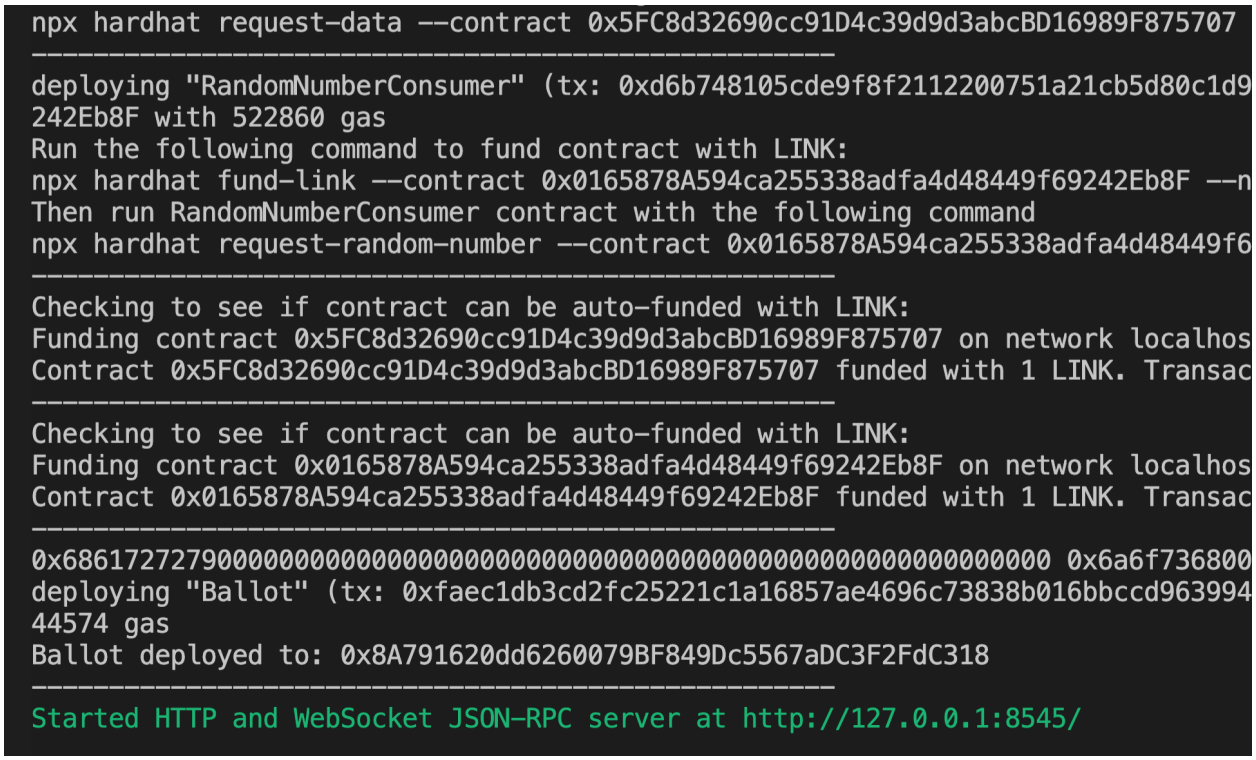
6. Now you can start your local hardhat network. You should run it separately in a terminal window, so it can also be referenced by other applications if needed. You can open a second terminal window by pressing the '+' icon near the bottom right of VS Code, so you can have one open for commands, and one for your hardhat local network. You can easily switch between the two.



Once you have a second terminal open, run the following command in the terminal

```
npx hardhat node
```

If you get any errors starting the node, please refer to possible solutions in the [Troubleshooting Appendix]



You're now ready to deploy the starter kit smart contracts to a local hardhat network!

Deploying and Interacting With the Smart Contracts

7. In your terminal in VS Code that **isn't** running the local hardhat node, run the following command to deploy the smart contracts in the hardhat starter kit to the local Hardhat network. You should see output similar to the following screenshot

```
npx hardhat deploy
```

```
pappas99@Harrys-MBP hardhat-starter-kit % npx hardhat deploy
Nothing to compile
Local network detected! Deploying mocks...
deploying "LinkToken" (tx: 0x41bdc5510e4107f9b41654ab795ec2e897ba033ffb689f80a2d48785247f08bc)...: deployed at 0x70e0bA845a1A0F2DA3359C97E0285013525FFC49
 with 1432927 gas
deploying "EthUsdAggregator" (tx: 0x3c690b2e4f7c3f8b3281af4b526436c97f944344fc98afd3f52daff4f2b2faed)...: deployed at 0x4826533B4897376654Bb4d4AD88B7faFD
0C98528 with 569671 gas
deploying "VRFCoordinatorMock" (tx: 0xc0289d51377d32bfbaa9b872914c8241973d98974ae1321b5b4ecdf86e133798)...: deployed at 0x99bbA657f2BbC93c02D617f8bA121cB
8Fc104Acf with 363662 gas
deploying "MockOracle" (tx: 0xa8889ecfed4ebcdde3fa2b68b7df32baebd17e71d0951026246430b111d0c5e7)...: deployed at 0x0E801D84Fa97b50751Dbf25036d067dCf18858b
F with 1131081 gas
Mocks Deployed!
----------------------------------------------------
You are deploying to a local network, you'll need a local network running to interact
Please run `npx hardhat console` to interact with the deployed smart contracts!
----------------------------------------------------
deploying "PriceConsumerV3" (tx: 0x8102721abcd1d18c20dd6a24a24679ad0e77a498d6958a0c15abd1020c17713b)...: deployed at 0x8f86403A4DE0BB5791fa46B8e795C54794
2fE4Cf with 156371 gas
Run Price Feed contract with command:
npx hardhat read-price-feed --contract 0x8f86403A4DE0BB5791fa46B8e795C547942fE4Cf --network localhost
----------------------------------------------------
```

All of the smart contracts in your project are now deployed to a simulated local Ethereum network running on your computer, and they're ready for you to start interacting with them. You'll notice that 'mock contracts' were deployed to simulate interacting with the Chainlink oracle network, this is because Chainlink doesn't know about your local network.

Price Feed Consumer Contract

1. To interact with the deployed Price Feed Consumer contract, run the 'read-price-feed' task, which will query the deployed smart contract, and return the latest price of the specified price feed. You can do this by running the following command, replacing the contract address with the one specified in your console output for the deployed PriceConsumerV3 contract. Alternatively, the console output during deployment gives you the exact command to run to query the PriceConsumerV3 contract, you can copy and paste the command directly from there

```
npx hardhat read-price-feed --contract
replace-with-your-PriceConsumerV3-address --network localhost
```

```
pappas99@Harrys-MBP hardhat-starter-kit % npx hardhat read-price-feed --contract 0xDc64a140Aa3E981100a9becA4E685f962f0cF6C9 --network localhost
Reading data from Price Feed consumer contract  0xDc64a140Aa3E981100a9becA4E685f962f0cF6C9  on network  localhost
Price is:  200000000000000000000
pappas99@Harrys-MBP hardhat-starter-kit %
```

As you can see, the value returned isn't the current price of ETH/USD, because in our '01_Deploy_PriceConsumerV3.js' deploy script, we deployed a mock aggregator contract to mimic an actual aggregator contract, and the value is set to a static amount.

Forking Ethereum Mainnet

1. Now that you've got a local Hardhat network working, let's modify it and change its initial state to be a fork of the current Ethereum mainnet. First thing you need to do, is modify

the hardhat.config.js file, and uncomment the line in the hardhat network section that mentions the MAINNET_RPC_URL. It should look like this

```
defaultNetwork: "hardhat",
    networks: {
        hardhat: {
            // If you want to do some forking, uncomment this
            forking: {
              url: MAINNET_RPC_URL
            }
        },
```

Before you start working with our local forked network, modify the Price Consumer deploy script, and the helper-hardhat-config.js file, so that we can reference the actual ETH/USD price feed contract that's currently on mainnet.

2. Open the file helper-hardhat-config.js, and add the 'ethUsdPriceFeed' config to the 31337 network . This is your locally running forked hardhat network. The value for this config will be '0x5f4eC3Df9cbd43714FE2740f5E3616155c5b8419', set to the contract address of the ETH/USD price feed on Ethereum mainnet, taken from The Chainlink Price Feeds Contract Documentation. Your config for the 31337 network should look like this in the file. Save the changes before continuing.

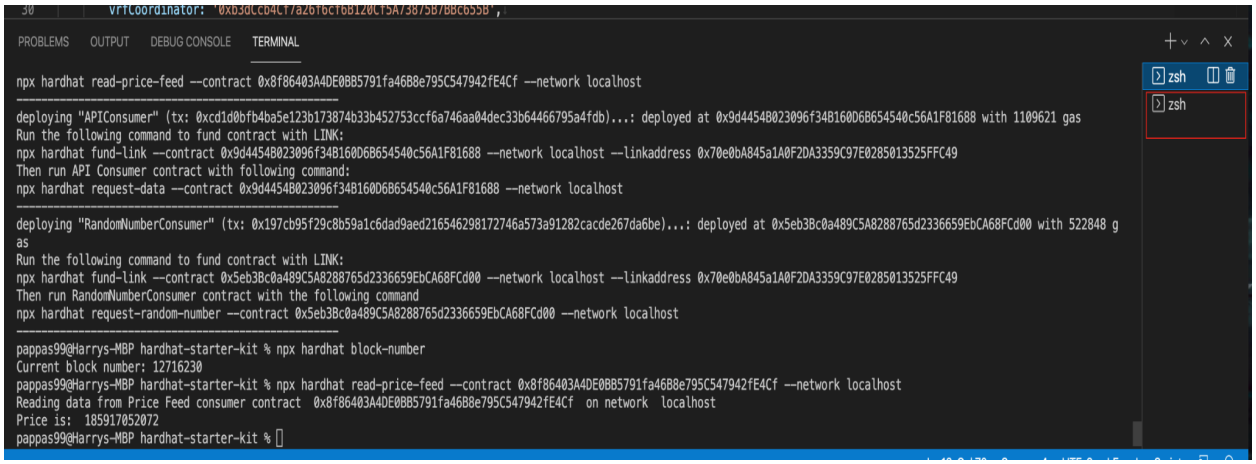
```
31337: {
        name: 'localhost',
        fee: '100000000000000000',
        keyHash:
'0x6c3699283bda56ad74f6b855546325b68d482e983852a7a82979cc4807b641f4',
        jobId: '29fa9aa13bf1468788b7cc4a500a45b8',
        ethUsdPriceFeed: '0x5f4eC3Df9cbd43714FE2740f5E3616155c5b8419'
    },
```

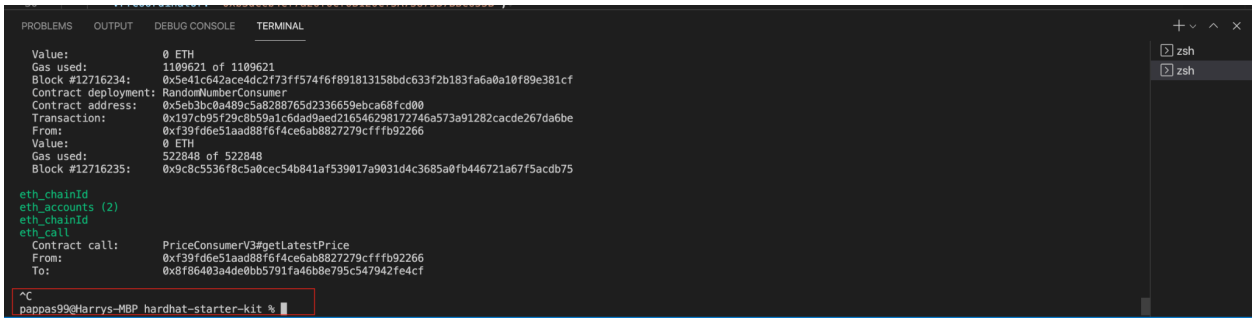
3. Next you need to modify your deployment script for the PriceConsumerV3 contract to use your newly added Price Feed config. Open the '01_Deploy_PriceConsumerV3.js' script from the 'deploy' folder. Find the logic in the script that checks to see if the chain ID is 31337, and if so to set the aggregator contract to the deployed mock one, and comment it out (add // at the beginning of the lines). You now want to always look up which aggregator contract to use from our config, even when working with a local network. This part of the script should look like the following. Changes are in bold

```
//if (chainId == 31337) {
        //const EthUsdAggregator = await
deployments.get('EthUsdAggregator')
       // ethUsdPriceFeedAddress = EthUsdAggregator.address
   // } else {
        ethUsdPriceFeedAddress =
networkConfig[chainId]['ethUsdPriceFeed']
   // }
```

You're now ready to deploy your smart contracts to a local forked network!

4. Now you need to go back to your VS Code terminal window, the one that's running the local hardhat node. You need to 'kill' and restart the process, so it can pick up the new config and fork mainnet. You can kill the currently running node by switching to the terminal currently running the hardhat node, and typing CTRL + C





Once the process has been killed, you can restart it using the following command (or press the up key to find it in your command history)

```
npx hardhat node
```

If you have issues starting the hardhat node, refer to the Troubleshooting Appendix for suggestions

5. Back in your other VS Code Terminal that **isn't** running the hardhat node, run the following command to deploy the smart contracts to the local hardhat network. Because of your modified hardhat.config.js file, it will automatically pick up the forked Ethereum mainnet, and use that as a starting point for the local network.

```
npx hardhat deploy
```

```
pappas99@Harrys-MBP hardhat-starter-kit % npx hardhat deploy
Nothing to compile
Local network detected! Deploying mocks...
deploying "LinkToken" (tx: 0x41bdc5510e4107f9b41654ab795ec2e897ba033ffb689f80a2d48785247f08bc)...: deployed at 0x70e0bA845a1A
7E0285013525FFC49 with 1432927 gas
deploying "EthUsdAggregator" (tx: 0x3c690b2e4f7c3f8b3281af4b526436c97f944344fc98afd3f52daff4f2b2faed)...: deployed at 0x48265
54Bb4d4AD88B7faFD0C98528 with 569671 gas
deploying "VRFCoordinatorMock" (tx: 0xc0289d51377d32bfbaa9b872914c8241973d98974ae1321b5b4ecdf86e133798)...: deployed at 0x99b
3c02D617f8bA121cB8Fc104Acf with 363662 gas
deploying "MockOracle" (tx: 0xa8889ecfed4ebcdde3fa2b68b7df32baebd17e71d0951026246430b111d0c5e7)...: deployed at 0x0E801D84Fa9
5036d067dCf18858bF with 1131081 gas
Mocks Deployed!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
You are deploying to a local network, you'll need a local network running to interact
Please run `npx hardhat console` to interact with the deployed smart contracts!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
deploying "PriceConsumerV3" (tx: 0x51d060ec338024ad454bf4f91d385aa84e1cdd1d22b0f2c2837591977ffdea23)...: deployed at 0x8f8640
1fa46B8e795C547942fE4Cf with 156371 gas
Run Price Feed contract with command:
npx hardhat read-price-feed --contract 0x8f86403A4DE0BB5791fa46B8e795C547942fE4Cf --network localhost
```

6. To test that we're actually working with a fork of mainnet, we can execute the 'block-number' task that comes included with the hardhat starter kit. This task can be found in the tasks folder (block-number.js), and simply returns the current block number in the blockchain.

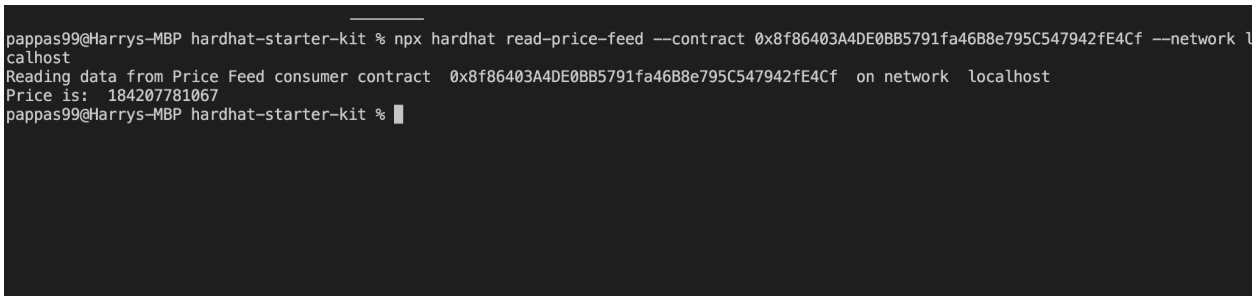
```
npx hardhat block-number --network localhost
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

pappas99@Harrys-MBP hardhat-starter-kit % npx hardhat block-number
Current block number: 12716017
pappas99@Harrys-MBP hardhat-starter-kit %
```

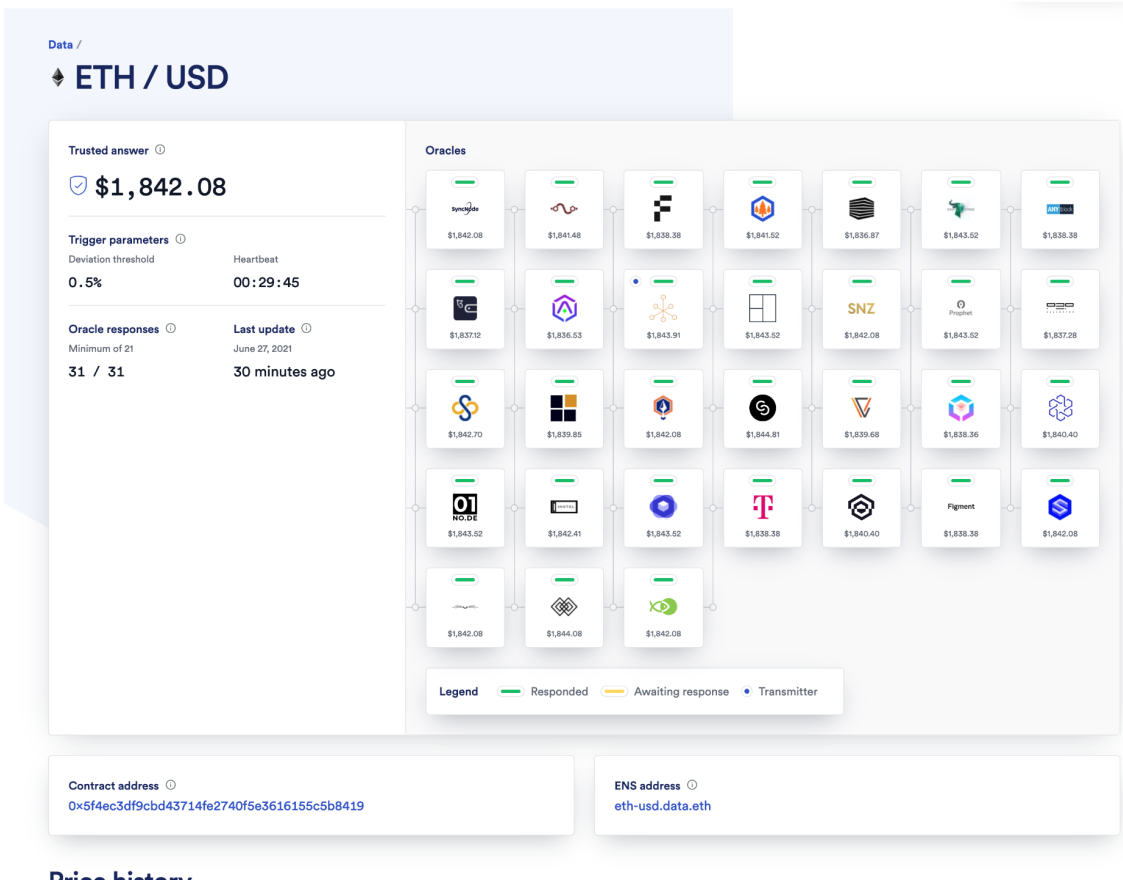
If it were a normal local hardhat network, the block number would be 0. You can compare the returned block number to the current mainnet block at https://etherscan.io/. The one at etherscan may be a few blocks ahead, by the time you deploy and then execute the block-number task etc. Now you're ready to execute the read-price-feed task to interact with the Price Feed consumer contract.

7. As per the output in the terminal when you deploy the smart contracts, run the command to execute the read-price-feed task, passing in the correct contract address as shown in the deploy output. The result should be that the returned price feed should be the price on the current Ethereum mainnet ETH/USD feed

```
npx hardhat read-price-feed --contract insert-contract-address-here
--network localhost
```



If you go to the page for the ETH/USD feed on Ethereum mainnet, you should see a matching result (or similar if a few blocks have passed)



Congratulations, you've successfully used Hardhat to perform the following:

1. Deploy smart contracts to a local network
2. Interacted with deployed smart contracts and mock contracts on a local network
3. Created a fork of the Ethereum mainnet on a local network, deployed smart contracts to it, and interacted with the mainnet ETH/USD price feed.

Bonus Exercises:

You can attempt to complete these exercises if you've completed the main exercise ahead of schedule:

Read the Mainnet Forking documentation in the hardat documentation, and try to fork mainnet at a specific block back in time, either by adding in the 'blockNumber' property into your hardhat.config.js file, or via the 'fork-block-number' parameter when you start your forked hardhat node. .

```
npx hardhat node --fork https://eth-mainnet.alchemyapi.io/v2/<key>
--fork-block-number 11095000
```

Run the block-number task each time you fork at a different block to ensure the chain has been forked at the right place, then execute the ‘read-price-feed’ task to see what the price of ETH/USD was in the Chainlink ETH/USD price feed at the time.

Exercise 4: Testing Smart Contracts

In this exercise, you’ll execute the unit and integration tests that come with the Hardhat Starter Kit, then do a Solidity Coverage test.

Installing Yarn

1. For the tests, instead of using the package manager NPM, we’ll switch and use Yarn, another popular package manager that’s faster than NPM. To install Yarn, run the following command

```
npm install -g yarn
```

If you have issues installing yarn, please refer to possible solutions in the Troubleshooting Appendix

2. You can verify the installation by running the following

```
yarn --version
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

pappas99@Harrys-MBP chainlink-hardhat-box-1 % yarn --version
1.22.10
pappas99@Harrys-MBP chainlink-hardhat-box-1 %
```

Executing the Unit Tests

1. Ensure your Hardhat Starter Kit project is still using your local Hardhat network as the currently running network, and that the local network is currently running in a terminal window. If not, follow steps from the last exercise to configure Hardhat and start the hardhat node

2. Now you're going to execute the unit tests. These will run a few basic tests against the smart contracts. Run the following command in the console to execute the unit tests

```
yarn test
```

```
pappas99@Harrys-MBP chainlink-hardhat-box-1 % npm test

> chainlink-hardhat-box@1.0.0 test
> hardhat test test/unit/PriceConsumerV3_unit_test.js --network hardhat; hardhat test test/unit/APIConsumer_unit_test.js --network hardhat; hardha
network hardhat


  PriceConsumer Unit Tests
Price Feed Value:  3311379522612620000
    ✓ should return a positive value (478ms)


  1 passing (2s)


  APIConsumer Unit Tests
Checking to see if contract can be auto-funded with LINK:
Funding contract 0xDc64a140Aa3E981100a9becA4E685f962f0cF6C9 on network hardhat
Contract 0xDc64a140Aa3E981100a9becA4E685f962f0cF6C9 funded with 1 LINK. Transaction Hash: 0xe539c918915353c7eb9a1b16d5b448c736ee7b408baf5eb1c68fbb
requestId:  0x1bfce59c2e0d7e0f015eb02ec4e04de4e67a1fe1508a4420cfd49c650758abe6
    ✓ Should successfully make an API request (138ms)
```

You should see all unit tests pass successfully. Next run the integration tests to test the actual integration of our smart contracts to the Chainlink Oracle network, but first you need to switch your hardhat config again to use the public Kovan test network.

Executing the Integration Tests

1. Back in the hardhat.config.js file, change the defaultNetwork to be Kovan

```
defaultNetwork: "kovan",
```

2. Next you need to revert the changes you made earlier when deploying the PriceConsumer contract. Open the '01_Deploy_PriceConsumerV3.js', and revert the changes you made earlier (ie, remove the comment slashes in the if/else statement (or you can use the undo command if you still had the code open from earlier). The if/else statement should look like this. Save your work.

```
    if (chainId == 31337) {
        const EthUsdAggregator = await
deployments.get('EthUsdAggregator')
        ethUsdPriceFeedAddress = EthUsdAggregator.address
    } else {
        ethUsdPriceFeedAddress =
networkConfig[chainId]['ethUsdPriceFeed']
    }
```

3. Now that you're back on the Kovan network, you need to deploy your smart contracts again, by running the following command

```
npx hardhat deploy
```

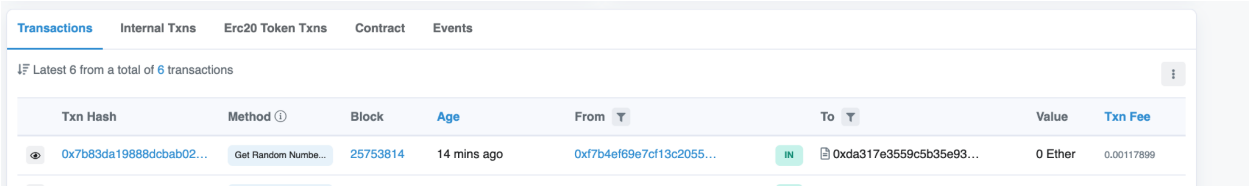
Once the deployment is complete, you can now execute the integration tests by running the following command using yarn

```
yarn test-integration
```

The integration tests may take a couple minutes to complete, because they have delays specified in them to give the Chainlink nodes enough time to perform the callback transactions.

```
pappas99@Harrys-MBP chainlink-hardhat-box-1 % yarn test-integration
yarn run v1.22.10
warning ../../package.json: No license field
$ hardhat test test/integration/APIConsumer_int_test.js --network kovan; hardhat test test/integration/RandomNumberConsumer_int_test.js --network kovan


  APIConsumer Integration Tests
API Consumer Volume:  337491423181095532070O
    ✓ Should successfully make an external API request and get a result (44083ms)


  1 passing (45s)


  RandomNumberConsumer Integration Tests
VRF Result:  33247407288224525000987441960734785355337226014655510314065412121 9
    ✓ Should successfully make a VRF request and get a result (50208ms)


  1 passing (51s)

✨  Done in 103.28s.
pappas99@Harrys-MBP chainlink-hardhat-box-1 %
```

You can take the contract address from the output of the contract deployment step earlier, and look them up on https://kovan.etherscan.io/ to see the transactions being performed as part of the integration tests



Checking the Solidity Coverage

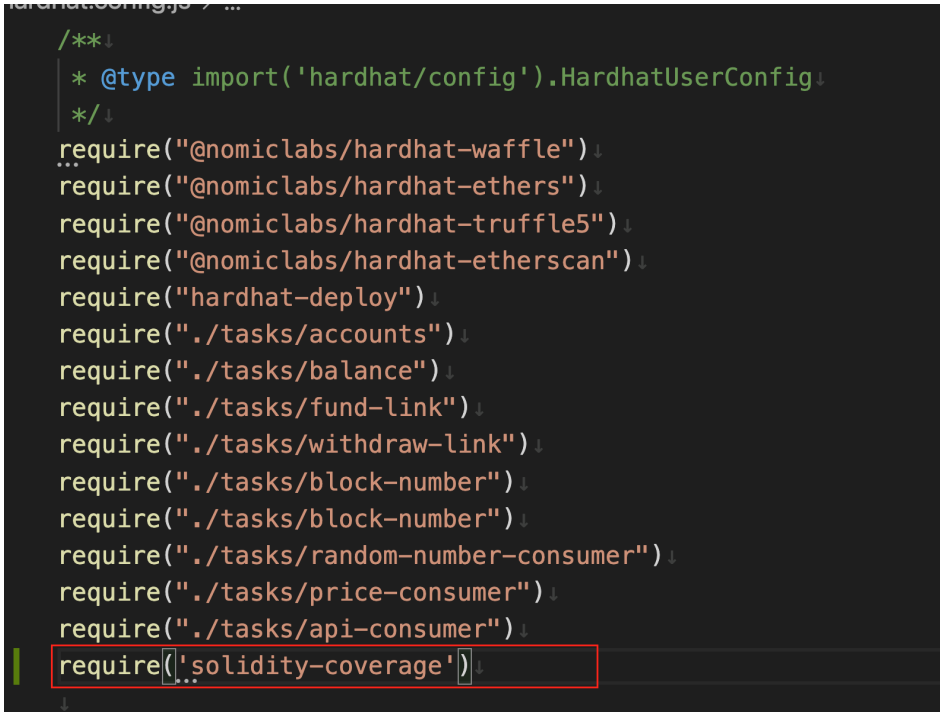
The last step in our testing is to check what the Solidity coverage is in our tests.

4. To start, you first need to install the solidity-coverage packages using the following command

```
npm install --save-dev solidity-coverage
```

5. Next, you need to add the following line to the top of your hardhat.config.js file, to let hardhat know about the library

```
require('solidity-coverage')
```



```
/**
 * @type import('hardhat/config').HardhatUserConfig
 */
require("@nomiclabs/hardhat-waffle")
require("@nomiclabs/hardhat-ethers")
require("@nomiclabs/hardhat-truffle5")
require("@nomiclabs/hardhat-etherscan")
require("hardhat-deploy")
require("./tasks/accounts")
require("./tasks/balance")
require("./tasks/fund-link")
require("./tasks/withdraw-link")
require("./tasks/block-number")
require("./tasks/block-number")
require("./tasks/random-number-consumer")
require("./tasks/price-consumer")
require("./tasks/api-consumer")
require('solidity-coverage')
```

6. In addition to this, also switch the default network back to ‘hardhat’

```
defaultNetwork: "hardhat",
```

If you previously stopped your local hardhat node, start it again in a terminal in VS Code with the command you previously started it with. If it’s still running, you can leave it.

```
npx hardhat node
```

7. Now you’re ready to run the solidity coverage. we’ll run it against the unit tests on the local hardhat network. This will check to see how complete the unit tests are

```
npx hardhat coverage --testfiles "test/unit/*.js"
```

```
pappas99@Harrys-MBP chainlink-hardhat-box-1 % npx hardhat coverage --testfiles "test/unit/*.js"

Version
=======
> solidity-coverage: v0.7.16

Instrumenting for coverage...
=============================

> APIConsumer.sol
> PriceConsumerV3.sol
> RandomNumberConsumer.sol
> test/LinkToken.sol
> test/MockOracle.sol
> test/MockV3Aggregator.sol
> test/VRFCoordinatorMock.sol

Compilation:
============

Nothing to compile
```

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	74.07	25	63.64	75	
APIConsumer.sol	72.22	33.33	60	73.68	22,70,78,79,85
PriceConsumerV3.sol	100	100	100	100	
RandomNumberConsumer.sol	66.67	0	50	66.67	44,54
contracts/test/	47.06	16.67	57.14	42.86	
LinkToken.sol	100	100	100	100	
MockOracle.sol	47.06	16.67	57.14	42.86	... 156,180,181
MockV3Aggregator.sol	100	100	100	100	
VRFCoordinatorMock.sol	100	100	100	100	
All files	63.64	20	61.11	61.22	

You can see the report for each contract, and see how much of the code is being tested.

Congratulations, you've learned how to install and use the Yarn package manager, how to execute unit tests on a local hardhat network, how to execute integration tests on a public network, and how to check the coverage of your unit tests. Now the last step is checking your project into a public GitHub repository so you can share it with the world!

Checking in your Project To a Public Code Repository

Perform the following steps to check in your hardhat project into a public repository for everyone else to see your work

1. **WARNING**! If you directly pasted in your private key inside your hardhat.config.js file from previous exercises due to an issue where you couldn't deploy your

contracts, it will be checked into a public repository. You should remove any references to your private key in your code before proceeding (apart from the .env file). Here is a way to try to get around the issue. If you didn't have this issue, and only have the private key in your .env file, please proceed to Step 6

2. Open up your hardhat.config.js file, and ensure the Kovan section under networks is as follows:

```
kovan: {
            url: KOVAN_RPC_URL,
             accounts: [PRIVATE_KEY],
            //accounts: {
            //    mnemonic: MNEMONIC,
            //},
            saveDeployments: true,
        },
```

3. Go to your .env file, and copy your private key string, then type the following command into your terminal and press enter

```
export PRIVATE_KEY='insert-private-key-string-here'
```

4. You can test if the environment variable has been set by running the following, you should see your private key displayed.

```
echo $PRIVATE_KEY
```

5. Repeat the step with your KOVAN_RPC_URL variable. Export it to the terminal

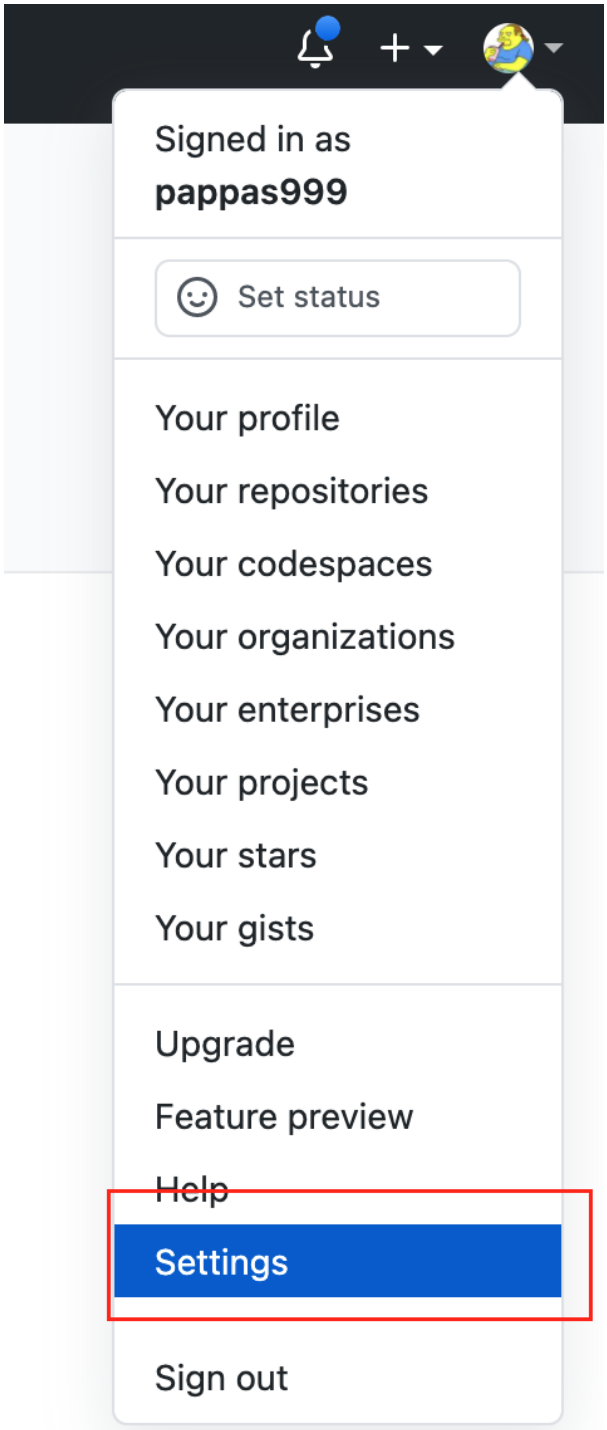
```
export
KOVAN_RPC_URL='https://kovan.infura.io/v3/insert-project-id-her
e'
```

```
echo $KOVAN_RPC_URL
```

Your project should now be safe to check into GitHub. You can test that the set environment variables work by running a deployment of your Smart Contracts to Kovan

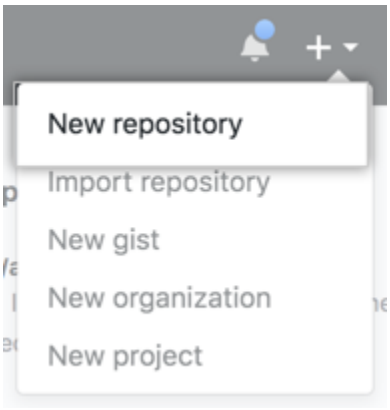
```
npx hardhat deploy --network kovan
```

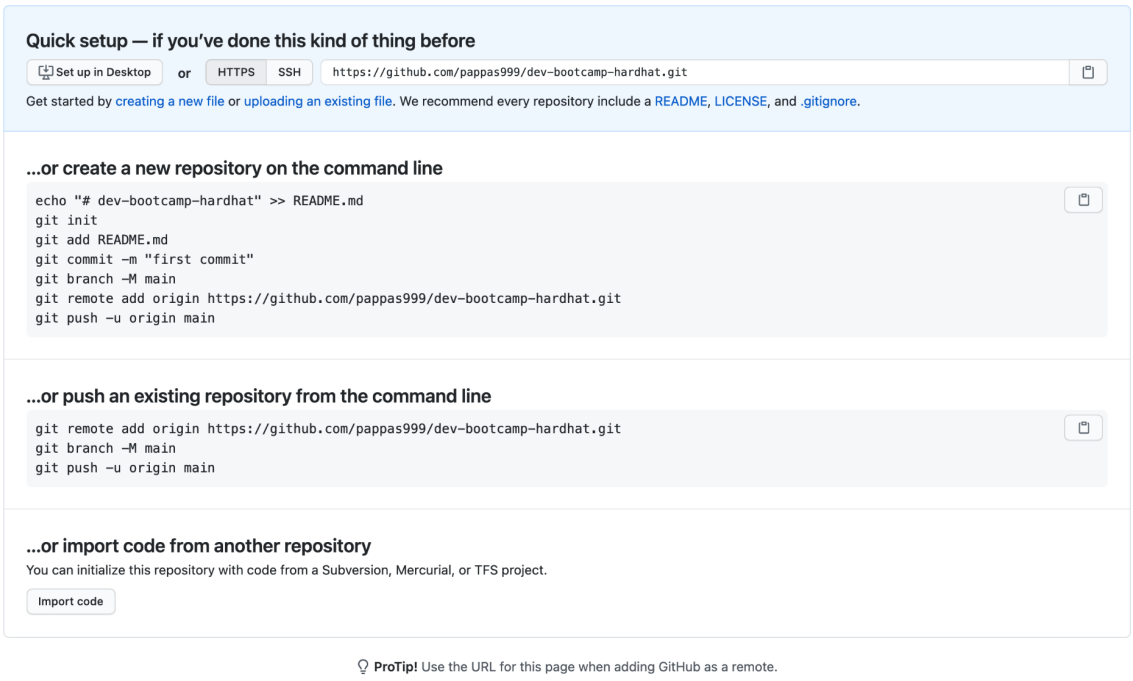
6. Head over to https://github.com/ and sign-in, or create a new free account if you don't have one already.
   Click on your profile on the top-right corner, and goto Settings -> Developer Settings -> Personal access tokens



7. Generate a personal access token as per the GitHub instructions

8. Create a new repository on GitHub. Call it ‘dev-bootcamp-hardhat’, and leave the visibility as ‘public’. To avoid errors, do not initialize the new repository with README, license, or gitignore files. Once you get to the ‘setup page’, leave it open, you will come back to it soon.





9. Go back to your VS Code terminal for your completed project

10. Initialize the local directory as a Git repository.

```
git init -b main
```

11. Add the files in your new local repository. This stages them for the first commit.

```
git add .
```

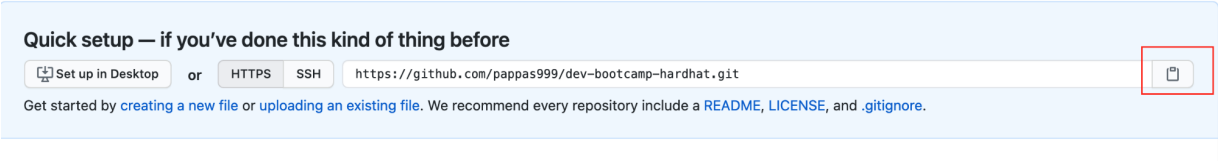
12. Next we want to remove the existing remote link to the Hardhat Starter Kit repository on GitHub

```
git remote rm origin
```

13. Commit the files that you've staged in your local repository.

```
git commit -m "Developer Bootcamp".
```

14. Go back to GitHub. At the top of your GitHub repository's Quick Setup page, click the copy button to copy the remote repository URL.



15. Back in VS code terminal, you need to add the URL for the remote repository where your local repository will be pushed. Enter in the command below, and replace the <PASTE_REMOTE_URL> with the value you copied above. If prompted for GitHub authentication details, use your email that you signed up to GitHub with, and your personal access token that you generated earlier as the password. If you don't get prompted, you can continue.

```
git remote add origin  <PASTE_REMOTE_URL>
```

16. Verify the new remote URL with the following command. You should see it now pointing to your GitHub repository. This means when we push code, we will be pushing it to your GitHub repository.

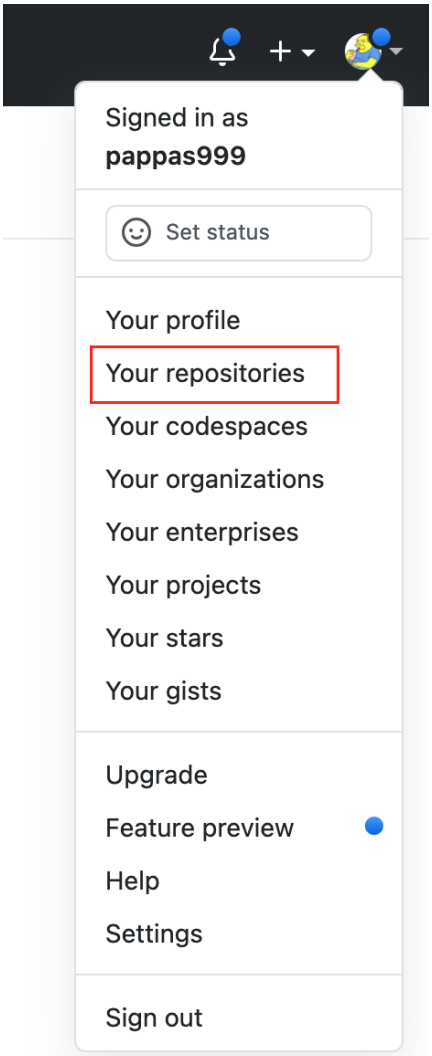
```
git remote -v
```

```
pappas99@Harrys-MBP hardhat-starter-kit % git remote -v
origin  https://github.com/pappas999/scdb2.git (fetch)
origin  https://github.com/pappas999/scdb2.git (push)
```

17. Push the changes in your local repository up to GitHub. If prompted for GitHub authentication details, use your email that you signed up to GitHub with, and your personal access token that you generated earlier as the password. This will give your local git program access to push code up to your GitHub account.

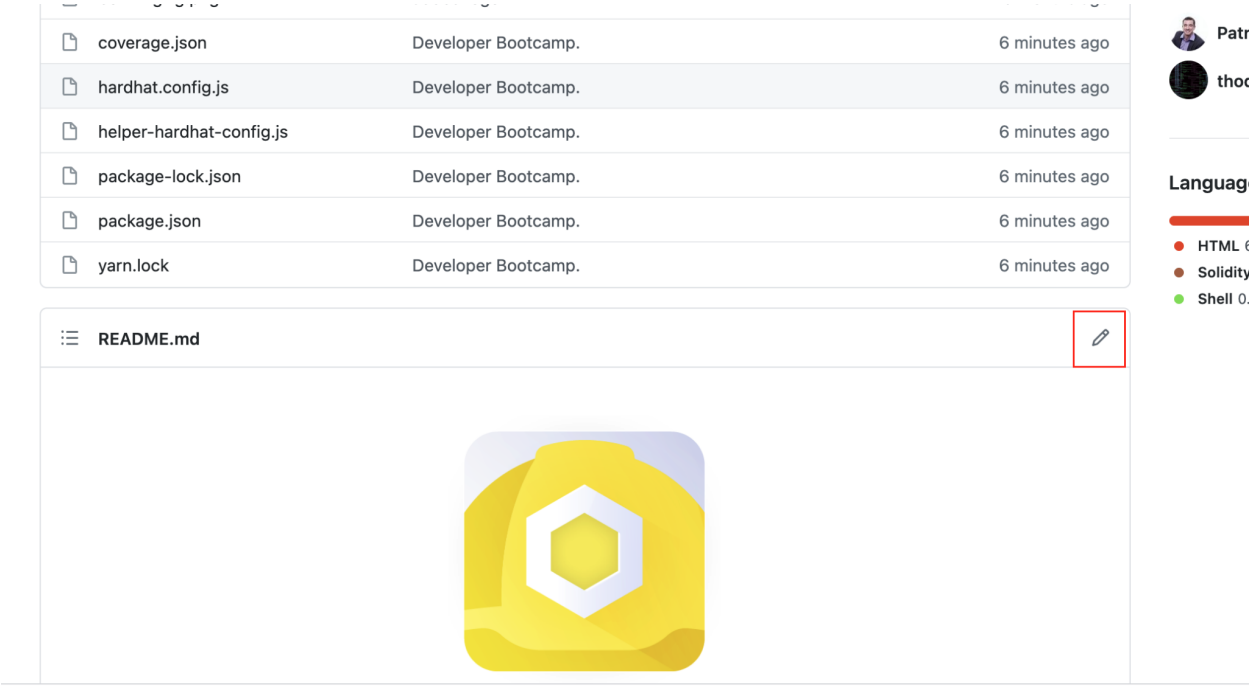
```
git push -u origin main
```

Your repository should now be live on GitHub! If you still have GitHub open with your new repository, then you can just refresh the page. Otherwise, head to GitHub, then in the top right corner open the menu and choose ‘your repositories’, then click on the URL link to your project to open it up and see. Take note of the URL and share it in the Bootcamp chat to share your completed project with everyone!



If you've made it this far, congratulations, you've completed the Summer 2021 Smart Contract Developer Bootcamp!

If you finished early, you can edit the README file and modify the text to something more appropriate. You can do this directly in GitHub by pressing on the edit icon, then modifying the file and saving your changes. Or you can modify the README file in your VS Code project, then repeat the steps to commit the change and upload them to GitHub. You can also edit the project about/description directly in GitHub.



Bonus Exercise:

You can attempt to complete these exercises if you've completed the main exercise ahead of schedule:

1. If you haven't completed the bonus exercises from earlier, create a new contract in your contracts folder called 'ERC-20.sol', paste the following code in it and save your work. If you have already this, you can skip this step

```
pragma solidity ^0.6.0;

interface IERC20 {

    function totalSupply() external view returns (uint256);
    function balanceOf(address account) external view returns
(uint256);
    function allowance(address owner, address spender) external
view returns (uint256);

    function transfer(address recipient, uint256 amount)
external returns (bool);
```

```
    function approve(address spender, uint256 amount) external
returns (bool);
    function transferFrom(address sender, address recipient,
uint256 amount) external returns (bool);


    event Transfer(address indexed from, address indexed to,
uint256 value);
    event Approval(address indexed owner, address indexed
spender, uint256 value);
}


contract ERC20Basic is IERC20 {

    string public constant name = "ERC20Basic";
    string public constant symbol = "ERC";
    uint8 public constant decimals = 18;


    //event Approval(address indexed tokenOwner, address
indexed spender, uint tokens);
    //event Transfer(address indexed from, address indexed to,
uint tokens);


    mapping(address => uint256) balances;

    mapping(address => mapping (address => uint256)) allowed;

    uint256 totalSupply_;

    using SafeMath for uint256;


   constructor(uint256 total) public {
    totalSupply_ = total;
    balances[msg.sender] = totalSupply_;
    }

    function totalSupply() public override view returns
(uint256) {
    return totalSupply_;
    }
```

```
    function balanceOf(address tokenOwner) public override view
returns (uint256) {
        return balances[tokenOwner];
    }

    function transfer(address receiver, uint256 numTokens)
public override returns (bool) {
        require(numTokens <= balances[msg.sender]);
        balances[msg.sender] =
balances[msg.sender].sub(numTokens);
        balances[receiver] = balances[receiver].add(numTokens);
        emit Transfer(msg.sender, receiver, numTokens);
        return true;
    }

    function approve(address delegate, uint256 numTokens)
public override returns (bool) {
        allowed[msg.sender][delegate] = numTokens;
        emit Approval(msg.sender, delegate, numTokens);
        return true;
    }

    function allowance(address owner, address delegate) public
override view returns (uint) {
        return allowed[owner][delegate];
    }

    function transferFrom(address owner, address buyer, uint256
numTokens) public override returns (bool) {
        require(numTokens <= balances[owner]);
        require(numTokens <= allowed[owner][msg.sender]);

        balances[owner] = balances[owner].sub(numTokens);
        allowed[owner][msg.sender] =
allowed[owner][msg.sender].sub(numTokens);
        balances[buyer] = balances[buyer].add(numTokens);
        emit Transfer(owner, buyer, numTokens);
        return true;
    }
}

library SafeMath {
    function sub(uint256 a, uint256 b) internal pure returns
(uint256) {
      assert(b <= a);
```

```
      return a - b;
    }

    function add(uint256 a, uint256 b) internal pure returns
(uint256) {
      uint256 c = a + b;
      assert(c >= a);
      return c;
    }
}
```

2. In your deploy folder, create a new script called ‘06_deploy_erc-20.js’. Paste the following deployment code in it, and save your work.

```
let { networkConfig} = require('../helper-hardhat-config')

module.exports = async ({
  getNamedAccounts,
  deployments
}) => {
  const { deploy, log, get } = deployments
  const { deployer } = await getNamedAccounts()
  const chainId = await getChainId()
  let linkTokenAddress
  let oracle
  let additionalMessage = ""
  //set log level to ignore non errors

ethers.utils.Logger.setLogLevel(ethers.utils.Logger.levels.ERRO
R)
  let totalSupply = 100000000

  const erc20 = await deploy('ERC20Basic', {
    from: deployer,
    args: [totalSupply],
    log: true
  })

  log("ERC-20 deployed to: " + erc20.address)
  log("----------------------------------------------------")
}
module.exports.tags = ['all', 'bonus']
```

3. Now you can compile and deploy your contracts again to your local hardhat network. You can specify which deployment scripts you want to run by specifying a tag. Our deploy script has been tagged as ‘bonus’ (see the bold text in the code above), so it will just run deploy scripts with that tag

```
npx hardhat compile
```

```
npx hardhat deploy --tags bonus
```

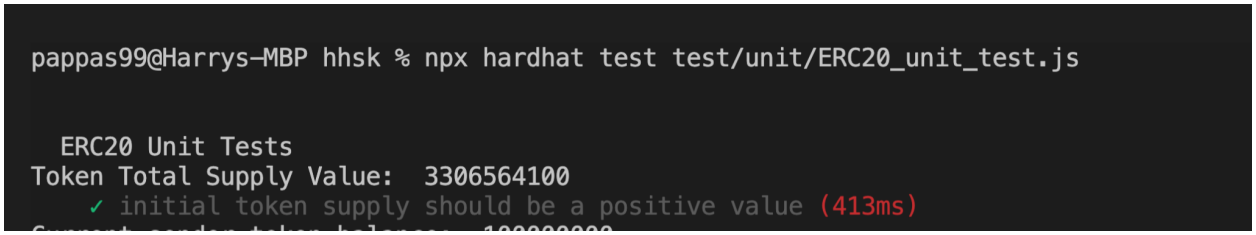
4. In the test/unit folder, create a new unit test script, call it ‘ERC20_unit_test.js’. Enter the following unit test into the script. It will check to see if the initial total supply entered when the contract is instantiated is a positive number

```
const { expect } = require('chai')
const chai = require('chai')
const BN = require('bn.js')
const skipIf = require('mocha-skip-if')
chai.use(require('chai-bn')(BN))
const { deployments, getChainId } = require('hardhat')
const { networkConfig, developmentChains } =
require('../../helper-hardhat-config')

skip.if(!developmentChains.includes(network.name)).
  describe('ERC20 Unit Tests', async function () {
    let erc20

    beforeEach(async () => {
      await deployments.fixture(['bonus'])
      const ERC20Basic = await deployments.get("ERC20Basic")
      erc20 = await ethers.getContractAt("ERC20Basic",
ERC20Basic.address)
    })

    it('initial token supply should be a positive value', async
() => {
      let result = await erc20.totalSupply()
```

```
      console.log("Token Total Supply Value: ", new
web3.utils.BN(result._hex).toString())
      expect(new
web3.utils.BN(result._hex).toString()).to.be.a.bignumber.that.i
s.greaterThan(new web3.utils.BN(0))
    })
  })
```

5. Execute the unit test with the following command

```
npx hardhat test test/unit/ERC20_unit_test.js
```



6. Try to add another unit test to the script. You can copy the ‘it’ statement already in there, and paste a new version of it underneath, changing details where required.

7. The new test description can be ‘ensure balance goes down after transfer of tokens’, and it should do the following:
   - Use your current ethereum address in your local hardhat node as the sender address, and your second ethereum address in your local hardhat node as the receiver address

```
        //first get current address information
        const accounts = await hre.ethers.getSigners();
        const account = await ethers.getSigners()
        const sender = account[0].address
        const receiver = account[1].address
```

   - store the balance of each account in a uint currentSenderBalance and currentReceiverBalance (hint: call the balanceOf function, passing in the addresses

```
        let currentSenderBalance = await
erc20.balanceOf(sender);
```

```
        let currentReceiverBalance = await
erc20.balanceOf(receiver);
        console.log("Current sender token balance: ",
currentSenderBalance.toString())
        console.log("Current receiver token balance: ",
currentReceiverBalance.toString())
```

- Call the ‘transfer’ function, sending some tokens from the sender to the receiver

```
        //do the transfer of 100 tokens from sender to receiver
        let transfer = await erc20.transfer(receiver,100)
```

- Store the new state of both accounts after the transfer into new variables

```
        //check both accounts
        let newSenderBalance = await erc20.balanceOf(sender);
        let newReceiverBalance = await
erc20.balanceOf(receiver);
        console.log("New sender token balance: ",
newSenderBalance.toString())
        console.log("New receiver token balance: ",
newReceiverBalance.toString())
```

- Do the assertion to ensure the new sender balance is less than the original balance

```
        //Run the assertion

expect(newSenderBalance.toString()).to.be.a.bignumber.that.is.l
essThan(currentSenderBalance.toString())
```

Your unit test is now ready to execute! Save your work and execute it with the following command

```
npx hardhat test test/unit/ERC20_unit_test.js
```

```
pappas99@Harrys-MBP hhsk % npx hardhat test test/unit/ERC20_unit_test.js


  ERC20 Unit Tests
Current sender token balance:  100000000
Current receiver token balance:  0
New sender token balance:  99999900
New receiver token balance:  100
    ✓ ensure balance goes down after transfer of tokens (62ms)


  1 passing (10s)
```

Appendix: Troubleshooting

Deploying compiled contracts

If you get an error similar to the following for one of your environment variables:

```
 * Invalid value undefined for HardhatConfig.networks.kovan.url
- Expected a value of type string.
```

Try manually exporting your KOVAN_RPC_URL variable via the following command in the terminal. Change the value to match what's in your .env file. You may need to repeat the step for your PRIVATE_KEY environment variable as well

MacOS and Linux commands:

```
export
KOVAN_RPC_URL='https://kovan.infura.io/v3/insert-key-here
```

```
export PRIVATE_KEY='insert-key-here'
```

Windows commands:

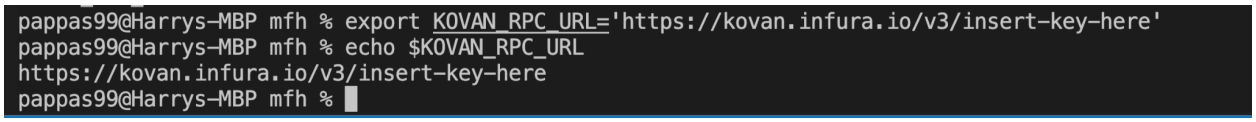
```
set KOVAN_RPC_URL='https://kovan.infura.io/v3/insert-key-here
```

```
set PRIVATE_KEY='insert-key-here'
```

Once this is done, you can check to see if the environment variables are set correctly by trying to echo them:

```
echo $KOVAN_RPC_URL
```

```
echo $PRIVATE_KEY
```



If you still can't get it working, if you're using windows, try adding the variable to your windows environment variables (in system settings), then try again

Starting the Local Hardhat Network

If you get an error similar to the following

```
Error in plugin hardhat-deploy:
Unsupported network for JSON-RPC server. Only hardhat is currently
supported.
hardhat-deploy cannot run on the hardhat provider when defaultNetwork is
not hardhat
```

Try and start the hardhat node with the following command

```
npx hardhat node --network hardhat
```

If you get an error similar to the following while attempting to start a Hardhat network that's a fork of Mainnet

```
Error HH604: Error running JSON-RPC server: Only absolute URLs
are supported
```

This means your fork URL is missing some components. Go to your .env file and ensure your variable is set correctly, and that the end hash string in your URL matches the key you obtained from Alchemy when you signed up during the setup instructions

```
ALCHEMY_MAINNET_RPC_URL='https://eth-mainnet.alchemyapi.io/v2/a
sdfsda8907s89df798sd7foshdfds9f8s7'
```

If you still have issues starting the mainnet fork hardhat node, try running it passing in the value of your ALCHEMY_MAINNET_RPC_URL on the command line. Ie exactly like this, replacing the 'insert-your-alchemy-key' string with the key you obtained from Alchemy

```
npx hardhat node --fork
https://eth-mainnet.alchemyapi.io/v2/insert-your-alchemy-key
```

If you are running Windows as none of the above work, try exiting VS Code and restarting it as Administrator (hold SHIFT, right click, Run As Administrator), then try starting the node again

Finally, if you still can't get the local node to start, try starting it outside of VS Code. i.e. open your terminal (macOS, linux) or Command Prompt (windows, ensure you start it as Administrator), navigate to your 'hardhat-starter-kit' directory and try starting the command to start the Hardhat node there. If successful, you can leave it running there and switch back to VS Code and still run the commands to deploy contracts etc as per normal in VS Code.

Installing Yarn

If you get an errors installing yarn:

Windows Users:

- Ensure you're running VS Code as Administrator (hold shift, right click, run as Administrator)
- Try installing it from the Command Prompt instead, ensuring you run the CMD.exe as Administrator
- Try downloading and installing it via the msi installer

macOS or Linux users

- Try running it with sudo in front of the command

```
sudo npm install -g yarn
```

If you're still stuck on installing yarn, check out some other possible installation methods on the Yarn installation page