LTIMindtree

Hello World Smart Contract

Table of Contents

1 Objective:

To Create Your First Solidity Smart Contract on Polygon Mumbai network.

2 Overview of this Tutorial

In this tutorial we will Create and deploy a Hello World smart contract on the Polygon Mumbai network using Metamask, Solidity, Hardhat and Alchemy.

Link of this tutorial is provided below-
How To Write a Solidity Smart Contract! - Hello World Pt 1 - YouTube

Note : Although in the video the smart contract is deployed on the Ropsten test network but The Ropsten test network is being deprecated by the Ethereum community. So we will deploy it on Polygon Mumbai Testnet.

3 Prerequisites:

3.1 Alchemy

Used to interact with the Polygon Proof-of-Stake (PoS) chain with the help of Polygon PoS API. Includes developer tooling to monitor requests and data analytics that demonstrate what happens under the hood during smart contract deployment.

3.2 Metamask

Virtual Cryptocurrency wallet used to store MATIC.

3.3 Solidity

Programming Language used to write smart contracts

3.4 HardHat

Development environment to compile, deploy, test and debug your Ethereum software. Used to build smart contracts locally before deploying to the live chain.

3.5 Ether.js

Library used to interact and make requests to Ethereum by wrapping standard JSON-RPC methods, used for contract deployment.

3.6 Visual Studio Code

Is a source code editor used to write the smart contracts.

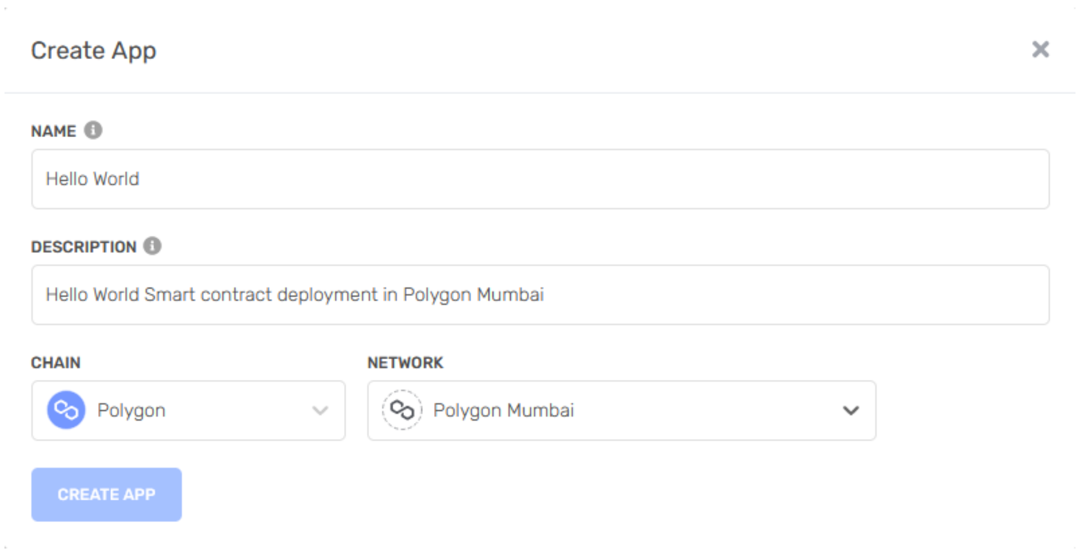
4 Steps to execute the tutorial:

4.1 Create a free Alchemy account:

Create your alchemy account by clicking on this link below
[Alchemy - Login]

4.2 Create an app on the Polygon chain on the Polygon Mumbai network.

After successfully creating an Alchemy account, API key is generated by creating an app. This authenticates the requests made to the Polygon Mumbai testnet.



Create App

NAME: Hello World

DESCRIPTION: Hello World Smart contract deployment in Polygon Mumbai

CHAIN: Polygon

NETWORK: Polygon Mumbai

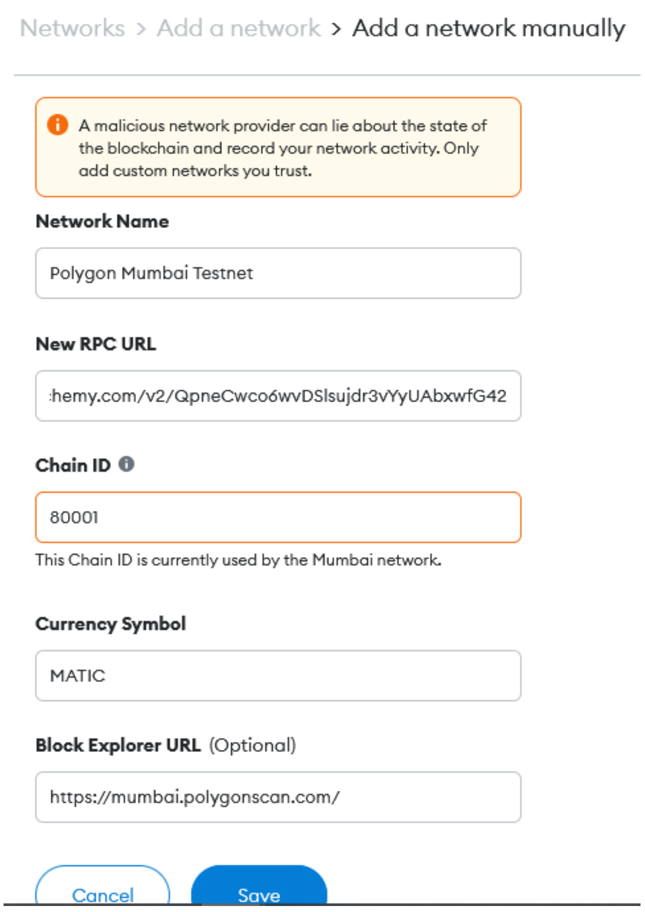
Figure 1 The Hello World app is created on the Polygon chain and the "Polygon Mumbai" testnet

4.3 Create a Metamask(Cryptocurrency wallet) account for the storage of the test MATIC.

Since Polygon PoS is a layer 2 scaling solution for Ethereum, we need to get an Ethereum wallet and add a custom Polygon URL to send and receive transactions on the Polygon Mumbai testnet. We will use Metamask, a browser-compatible digital wallet used to manage your wallet address.

Once Metamask is added as an extension:
1. Select “Settings” from the drop-down menu in the top right corner of your Metamask wallet.
2. Select “Networks” from the menu to the left.
3. Connect your wallet to the Mumbai Testnet using the following parameters.
   - Network Name: Polygon Mumbai Testnet
   - New RPC URL: https://polygon-mumbai.g.alchemy.com/v2/your-api-key
   - ChainID: 80001
   - Symbol: MATIC

- Block Explorer URL: https://mumbai.polygonscan.com/


Networks > Add a network > Add a network manually

A malicious network provider can lie about the state of the blockchain and record your network activity. Only add custom networks you trust.

Network Name: Polygon Mumbai Testnet

New RPC URL: :hemy.com/v2/QpneCwco6wvDSlsujdr3vYyUAbxwfG42

Chain ID: 80001

This Chain ID is currently used by the Mumbai network.

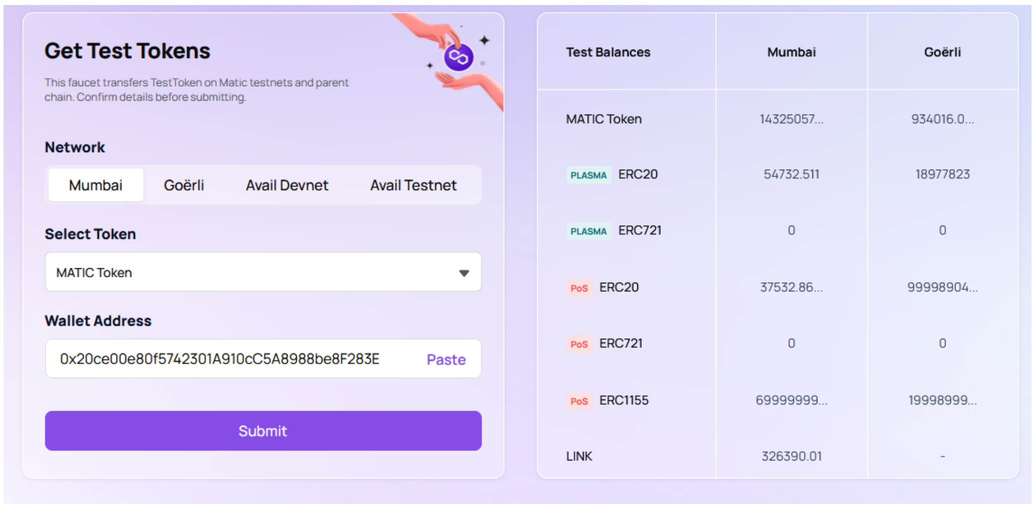
Currency Symbol: MATIC

Block Explorer URL (Optional): https://mumbai.polygonscan.com/

Figure 2 Adding Polygon Mumbai Testnet to Metamask wallet

4.4 Add Polygon Mumbai Test MATIC from a Faucet.

In order to deploy your smart contract to the test network, you need to obtain a few testnet tokens. To get testnet tokens, visit the Polygon Mumbai Faucet, select "Mumbai", choose "MATIC Token", and enter your Polygon wallet address, then click “Submit.” It may take some time to receive your testnet tokens due to network traffic.



Get Test Tokens

This faucet transfers TestToken on Matic testnets and parent chain. Confirm details before submitting.

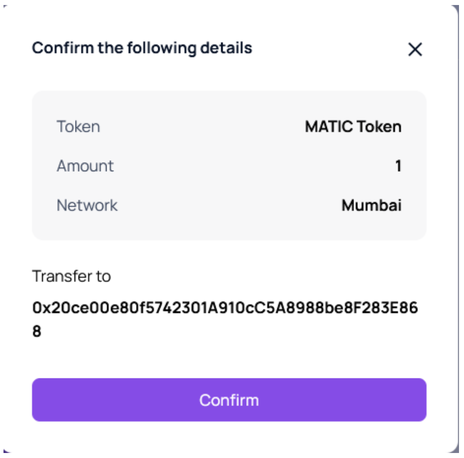
Network: Mumbai | Goërli | Avail Devnet | Avail Testnet

Select Token: MATIC Token

Wallet Address: 0x20ce00e80f5742301A910cC5A8988be8F283E

Test Balances	Mumbai	Goërli
MATIC Token	14325057...	934016.0...
PLASMA ERC20	54732.511	18977823
PLASMA ERC721	0	0
PoS ERC20	37532.86...	99998904...
PoS ERC721	0	0
PoS ERC1155	69999999...	19998999...
LINK	326390.01	-

Figure 3 Obtaining test MATIC from Polygon faucet



Confirm the following details

Token	MATIC Token
Amount	1
Network	Mumbai

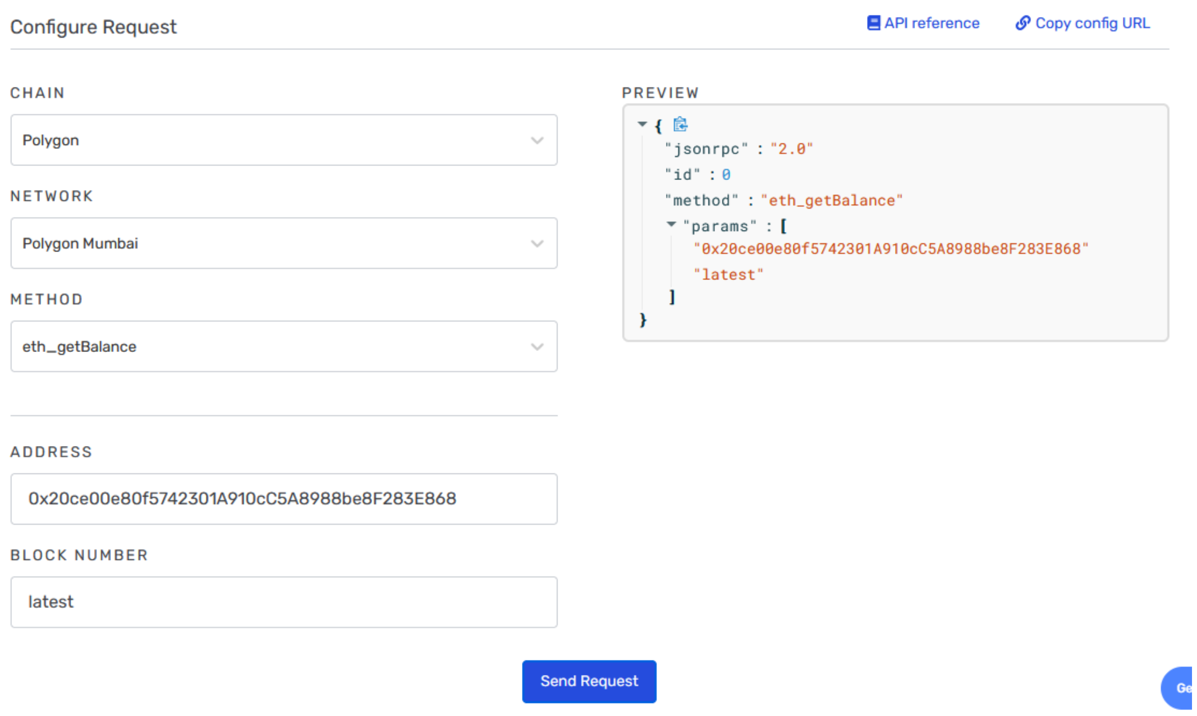
Transfer to

0x20ce00e80f5742301A910cC5A8988be8F283E86
8

Figure 4 Transaction details for the Transfer

4.5 Check Balance:

Check Balance with the help of eth_getBalance request using Alchemy’s composer tool. Select "Polygon" as the chain, "Polygon Mumbai" as the network, "eth_getBalance" as the method, and input your address. This will return the amount of MATIC in our wallet.



CHAIN: Polygon

NETWORK: Polygon Mumbai

METHOD: eth_getBalance

ADDRESS: 0x20ce00e80f5742301A910cC5A8988be8F283E868

BLOCK NUMBER: latest

PREVIEW

```
{
  "jsonrpc" : "2.0"
  "id" : 0
  "method" : "eth_getBalance"
  "params" : [
    "0x20ce00e80f5742301A910cC5A8988be8F283E868"
    "latest"
  ]
}
```

Figure 5 Request for getting the Balance of the Polygon wallet

4.6 Initialize the Project:

```
mkdir hello-world
cd hello-world
```

Figure 6 Creating a new folder for the project

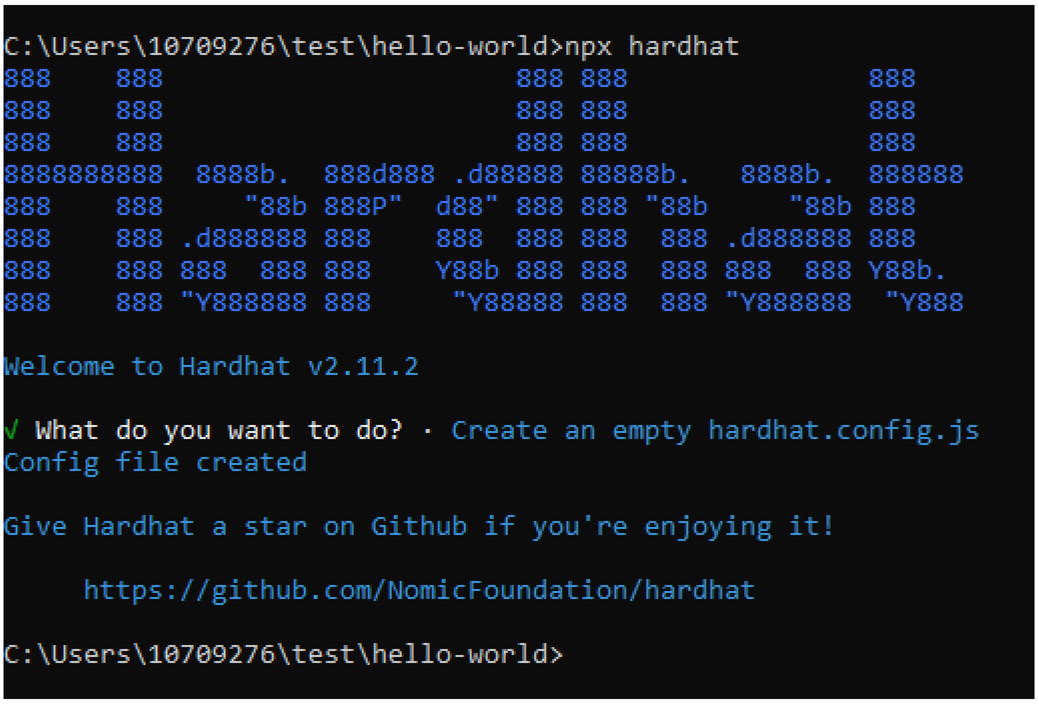
```
npm init # (or npm init --yes)
```

Figure 7 Initializing the project using npm

4.7 Download HardHat and create HardHat project:

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

Figure 8 Run this insider the hello-world project



```
C:\Users\10709276\test\hello-world>npx hardhat
888    888                      888 888               888
888    888                      888 888               888
888    888                      888 888               888
8888888888  8888b.  888d888 .d88888 88888b.   8888b.  888888
888    888     "88b 888P"  d88" 888 888 "88b     "88b 888
888    888 .d888888 888    888  888 888  888 .d888888 888
888    888 888  888 888    Y88b 888 888  888 888  888 Y88b.
888    888 "Y888888 888     "Y88888 888  888 "Y888888  "Y888

Welcome to Hardhat v2.11.2

√ What do you want to do? · Create an empty hardhat.config.js
Config file created

Give Hardhat a star on Github if you're enjoying it!

     https://github.com/NomicFoundation/hardhat

C:\Users\10709276\test\hello-world>
```

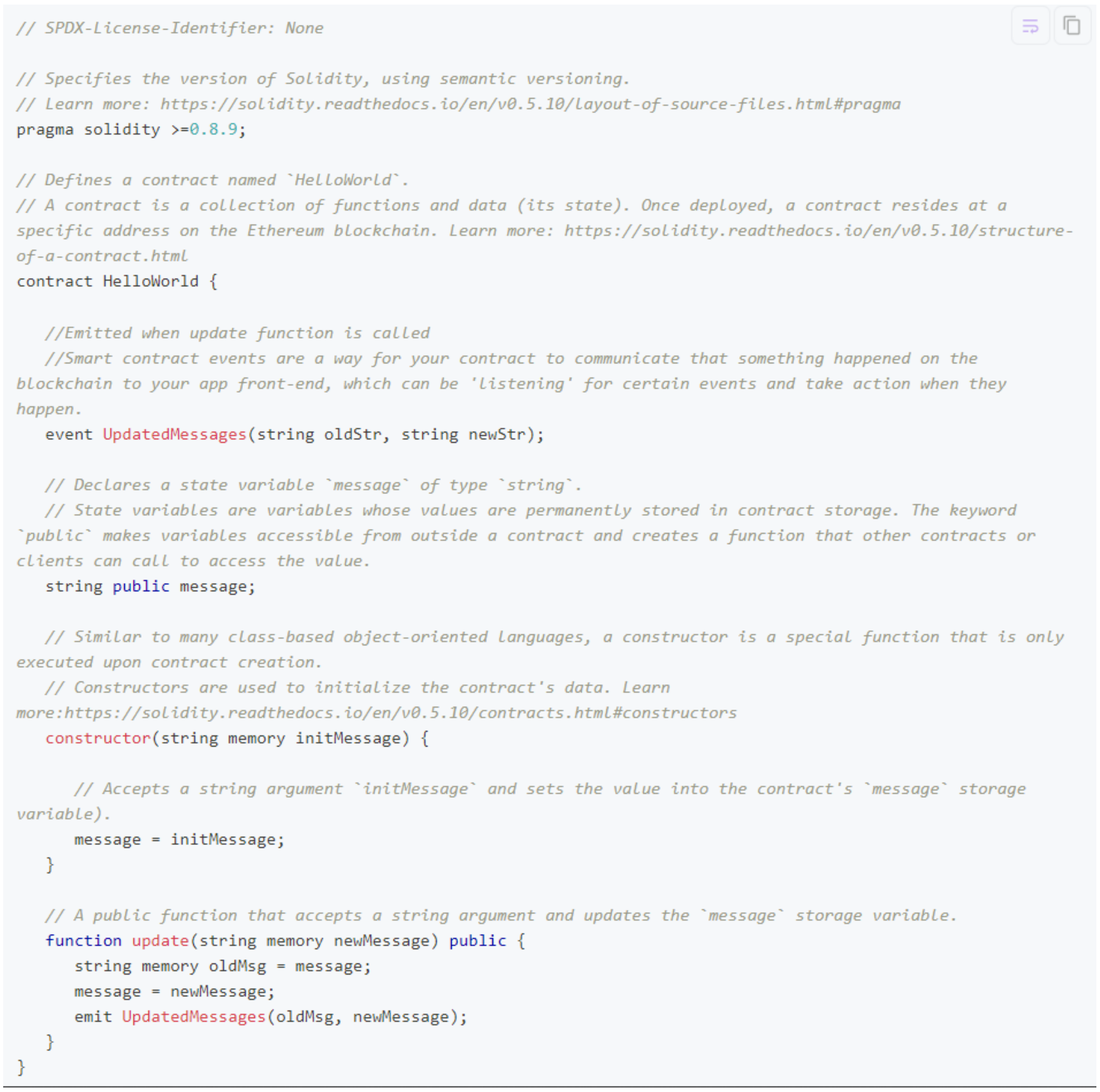
Figure 9 Creating a new Hardhat project

4.8 Add project folders

Create two new folders, **contracts** and **scripts** to keep the project organized.
contracts is where we will keep our hello world smart contract code file.
scripts is where we will keep scripts to deploy and interact with our contract.

4.9 Write the contract:

Open the hello world project in VSCode and write the HelloWorld.sol smart contract.
Navigate to the “contracts” folder and create a new file called HelloWorld.sol.



```
// SPDX-License-Identifier: None

// Specifies the version of Solidity, using semantic versioning.
// Learn more: https://solidity.readthedocs.io/en/v0.5.10/layout-of-source-files.html#pragma
pragma solidity >=0.8.9;

// Defines a contract named `HelloWorld`.
// A contract is a collection of functions and data (its state). Once deployed, a contract resides at a
specific address on the Ethereum blockchain. Learn more: https://solidity.readthedocs.io/en/v0.5.10/structure-
of-a-contract.html
contract HelloWorld {

   //Emitted when update function is called
   //Smart contract events are a way for your contract to communicate that something happened on the
blockchain to your app front-end, which can be 'listening' for certain events and take action when they
happen.
   event UpdatedMessages(string oldStr, string newStr);

   // Declares a state variable `message` of type `string`.
   // State variables are variables whose values are permanently stored in contract storage. The keyword
`public` makes variables accessible from outside a contract and creates a function that other contracts or
clients can call to access the value.
   string public message;

   // Similar to many class-based object-oriented languages, a constructor is a special function that is only
executed upon contract creation.
   // Constructors are used to initialize the contract's data. Learn
more:https://solidity.readthedocs.io/en/v0.5.10/contracts.html#constructors
   constructor(string memory initMessage) {

      // Accepts a string argument `initMessage` and sets the value into the contract's `message` storage
variable).
      message = initMessage;
   }

   // A public function that accepts a string argument and updates the `message` storage variable.
   function update(string memory newMessage) public {
      string memory oldMsg = message;
      message = newMessage;
      emit UpdatedMessages(oldMsg, newMessage);
   }
}
```

Figure 10 This is a super simple smart contract that stores a message upon creation and can be updated by calling the update function.

4.10 Connect Metamask & Alchemy to your project:

We've created a Metamask wallet, Alchemy account, and written our smart contract, now it's time to connect the three.
Every transaction sent from your virtual wallet requires a signature using your unique private key. To provide our program with this permission, we can safely store our private key (and Alchemy API key) in an environment file.

First, install the dotenv package in your project directory by:
npm install dotenv --save

Then, create a .env file in the root directory of our project, and add your Metamask private key and HTTP Alchemy API URL to it.

```
API_URL = "https://polygon-mumbai.g.alchemy.com/v2/your-api-key"
PRIVATE_KEY = "your-metamask-private-key"
```

Figure 11 The .env should look like this

4.11 Install Ether.js

```
npm install --save-dev @nomiclabs/hardhat-ethers "ethers@^5.0.0"
```

Figure 12 Command used to install ether.js

4.12 Update hardhat.config.js

Updating the several dependencies and plugins used so far in the hardhat.config.js.

```
/**
* @type import('hardhat/config').HardhatUserConfig
*/

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

const { API_URL, PRIVATE_KEY } = process.env;

module.exports = {
   solidity: "0.8.9",
   defaultNetwork: "polygon_mumbai",
   networks: {
      hardhat: {},
      polygon_mumbai: {
         url: API_URL,
         accounts: [`0x${PRIVATE_KEY}`]
      }
   },
}
```

Figure 13 The hardhat.config.js file should look like this

4.13 Compile our contract:

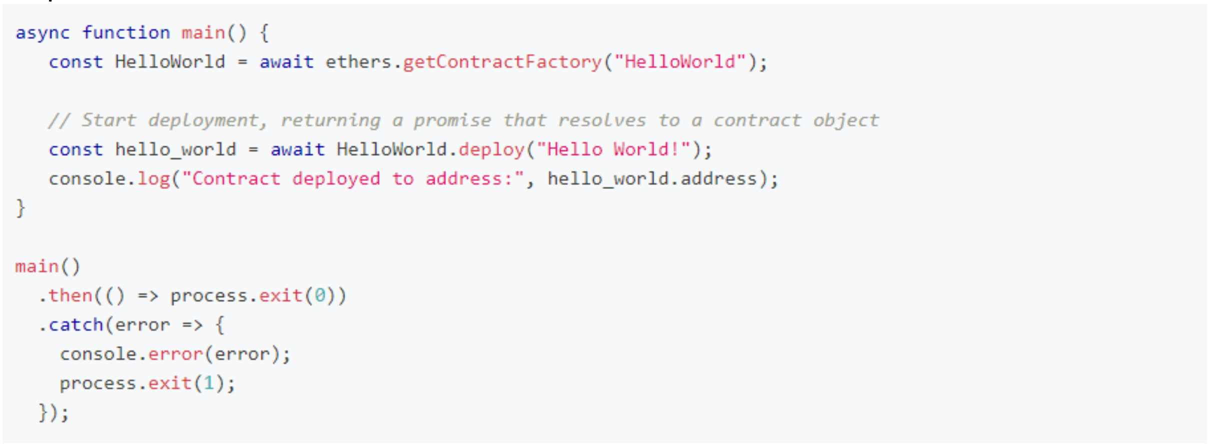
From the command line run:

```
npx hardhat compile
```

Figure 14 Compiling the project

4.14 Write the deploy script

Navigate to the scripts folder and create a new file called deploy.js which contains the contract deploy script.



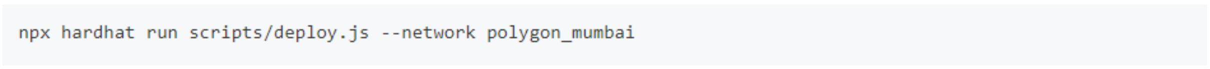
```
async function main() {
   const HelloWorld = await ethers.getContractFactory("HelloWorld");

   // Start deployment, returning a promise that resolves to a contract object
   const hello_world = await HelloWorld.deploy("Hello World!");
   console.log("Contract deployed to address:", hello_world.address);
}

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

Figure 15 deploy.js

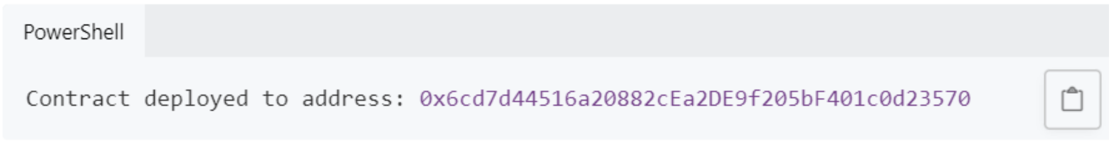
4.15 Deploy the contract:



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

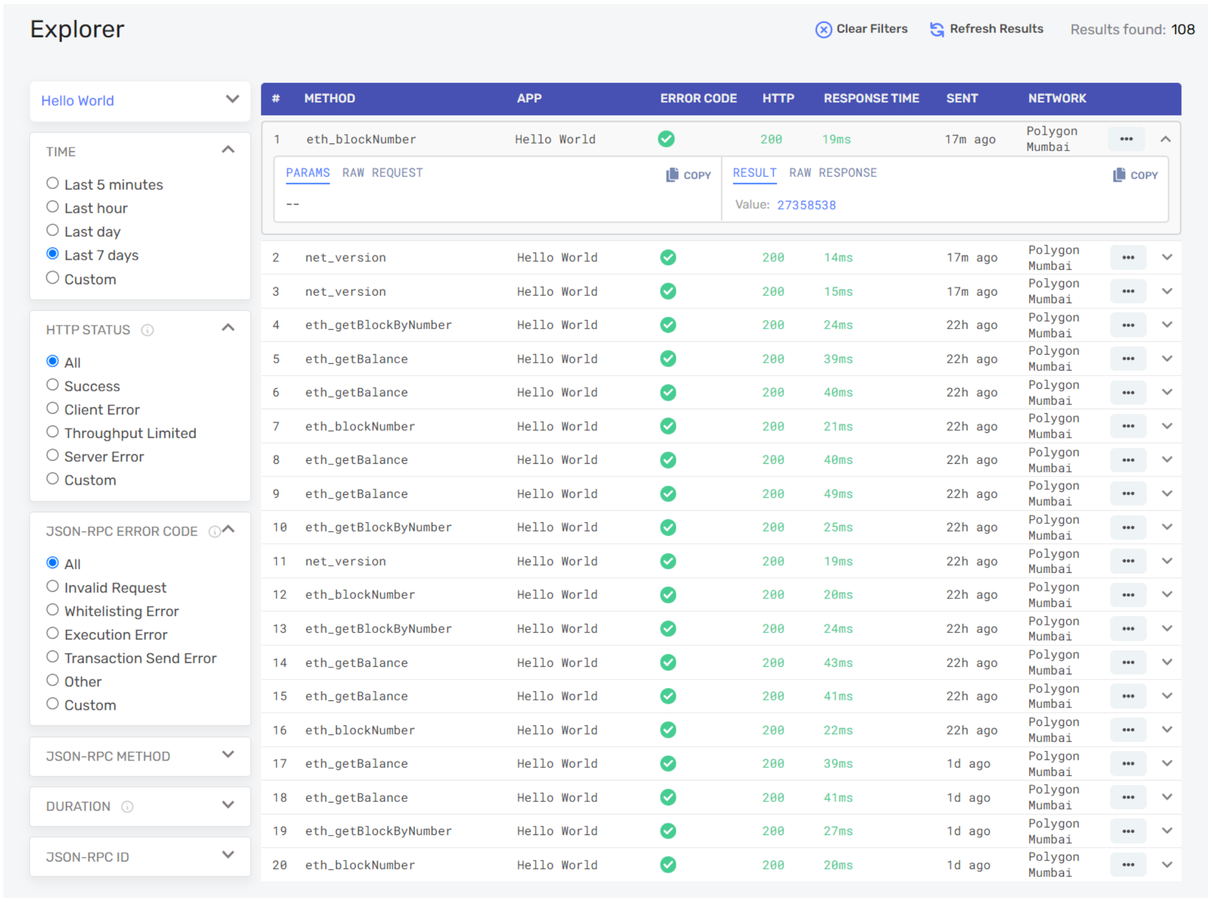
Figure 16 Command used to deploy the smart contract on the polygon mumbai network

You should then see something like:



```
Contract deployed to address: 0x6cd7d44516a20882cEa2DE9f205bF401c0d23570
```

4.16 Verify the contract



Explorer

Clear Filters Refresh Results Results found: 108

Hello World

TIME
- Last 5 minutes
- Last hour
- Last day
- [✓ Last 7 days]
- Custom

HTTP STATUS
- [✓ All]
- Success
- Client Error
- Throughput Limited
- Server Error
- Custom

JSON-RPC ERROR CODE
- [✓ All]
- Invalid Request
- Whitelisting Error
- Execution Error
- Transaction Send Error
- Other
- Custom

JSON-RPC METHOD

DURATION

JSON-RPC ID

#	METHOD	APP	ERROR CODE	HTTP	RESPONSE TIME	SENT	NETWORK
1	eth_blockNumber	Hello World	✓	200	19ms	17m ago	Polygon Mumbai
2	net_version	Hello World	✓	200	14ms	17m ago	Polygon Mumbai
3	net_version	Hello World	✓	200	15ms	17m ago	Polygon Mumbai
4	eth_getBlockByNumber	Hello World	✓	200	24ms	22h ago	Polygon Mumbai
5	eth_getBalance	Hello World	✓	200	39ms	22h ago	Polygon Mumbai
6	eth_getBalance	Hello World	✓	200	40ms	22h ago	Polygon Mumbai
7	eth_blockNumber	Hello World	✓	200	21ms	22h ago	Polygon Mumbai
8	eth_getBalance	Hello World	✓	200	40ms	22h ago	Polygon Mumbai
9	eth_getBalance	Hello World	✓	200	49ms	22h ago	Polygon Mumbai
10	eth_getBlockByNumber	Hello World	✓	200	25ms	22h ago	Polygon Mumbai
11	net_version	Hello World	✓	200	19ms	22h ago	Polygon Mumbai
12	eth_blockNumber	Hello World	✓	200	20ms	22h ago	Polygon Mumbai
13	eth_getBlockByNumber	Hello World	✓	200	24ms	22h ago	Polygon Mumbai
14	eth_getBalance	Hello World	✓	200	43ms	22h ago	Polygon Mumbai
15	eth_getBalance	Hello World	✓	200	41ms	22h ago	Polygon Mumbai
16	eth_blockNumber	Hello World	✓	200	22ms	22h ago	Polygon Mumbai
17	eth_getBalance	Hello World	✓	200	39ms	1d ago	Polygon Mumbai
18	eth_getBalance	Hello World	✓	200	41ms	1d ago	Polygon Mumbai
19	eth_getBlockByNumber	Hello World	✓	200	27ms	1d ago	Polygon Mumbai
20	eth_blockNumber	Hello World	✓	200	20ms	1d ago	Polygon Mumbai

Row 1 expanded:

PARAMS RAW REQUEST COPY

--

RESULT RAW RESPONSE COPY

Value: 27358538

Figure 17 Using Alchemy explorer to find information about the methods deployed along the smart contract.