Hello World Smart Contract

Objective:

Creating and deploying a Hello World smart contract with Alchemy on the Polygon Mumbai network using Metamask, Solidity, Hardhat and Alchemy.

Software/ Libraries used:

Alchemy 🡪 Used to interact with the Polygon 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.

Metamask 🡪 Virtual Cryptocurrency wallet used to store MATIC.

Solidity 🡪 Programming Language used to write smart contracts

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

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

Visual Studio Code 🡪 Is a source code editor used to write the smart contracts.

Steps:

1. Create a free Alchemy account:

Alchemy's developer platform can be used to interact from the Alchemy Polygon PoS API to communicate with the Polygon PoS chain. The platform includes developer tooling to monitor requests and data analytics that demonstrate what happens under the hood during smart contract deployment.

1. 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.

Graphical user interface, text, application

Description automatically generated

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

1. 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[​](https://wiki.polygon.technology/docs/develop/alchemy/#network-name-polygon-mumbai-testnet)
* New RPC URL: <https://polygon-mumbai.g.alchemy.com/v2/your-api-key>[​](https://wiki.polygon.technology/docs/develop/alchemy/#new-rpc-url-httpspolygon-mumbaigalchemycomv2your-api-key)
* ChainID: 80001[​](https://wiki.polygon.technology/docs/develop/alchemy/#chainid-80001)
* Symbol: MATIC[​](https://wiki.polygon.technology/docs/develop/alchemy/#symbol-matic)
* Block Explorer URL: <https://mumbai.polygonscan.com/>

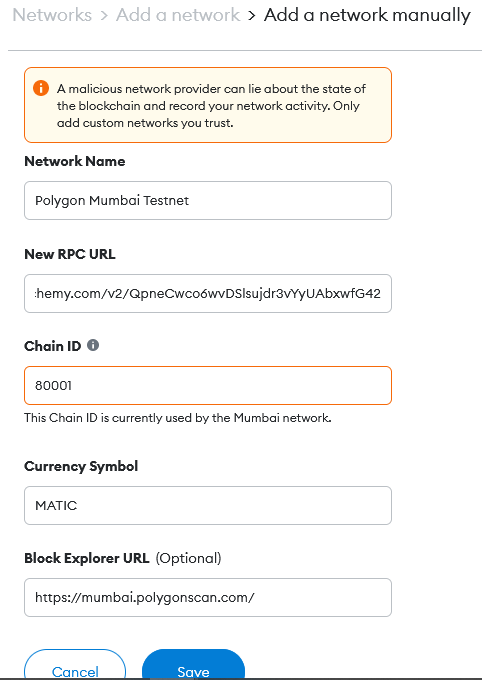


Figure 2 Adding Polygon Mumbai Testnet to Metamask wallet

1. 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](https://faucet.polygon.technology/), 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.

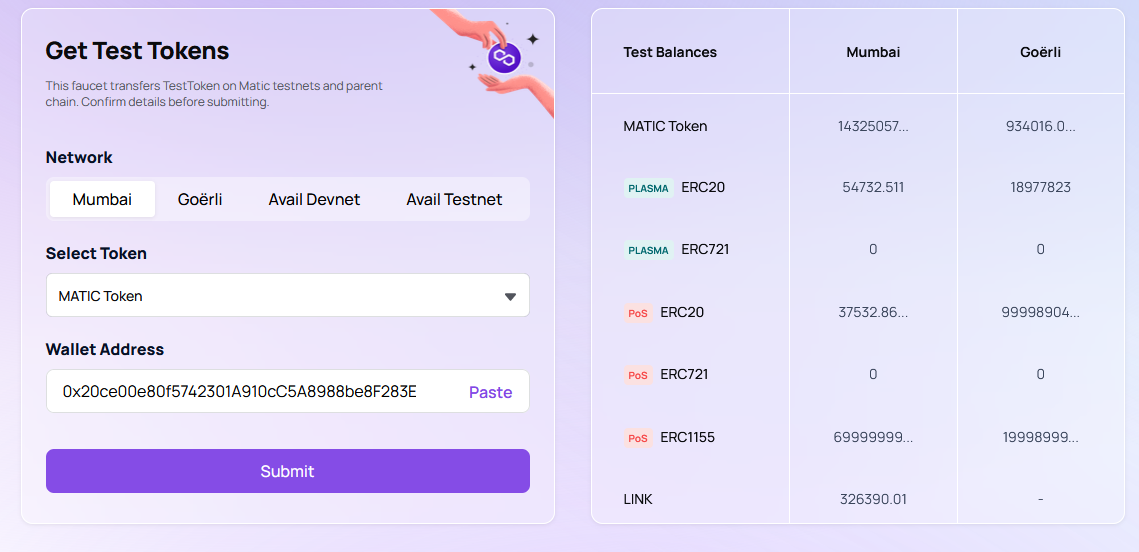


Figure 3 Obtaining test MATIC from Polygon faucet

Graphical user interface, text, application, Teams

Description automatically generated

Figure 4 Transaction details for the Transfer

1. 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.

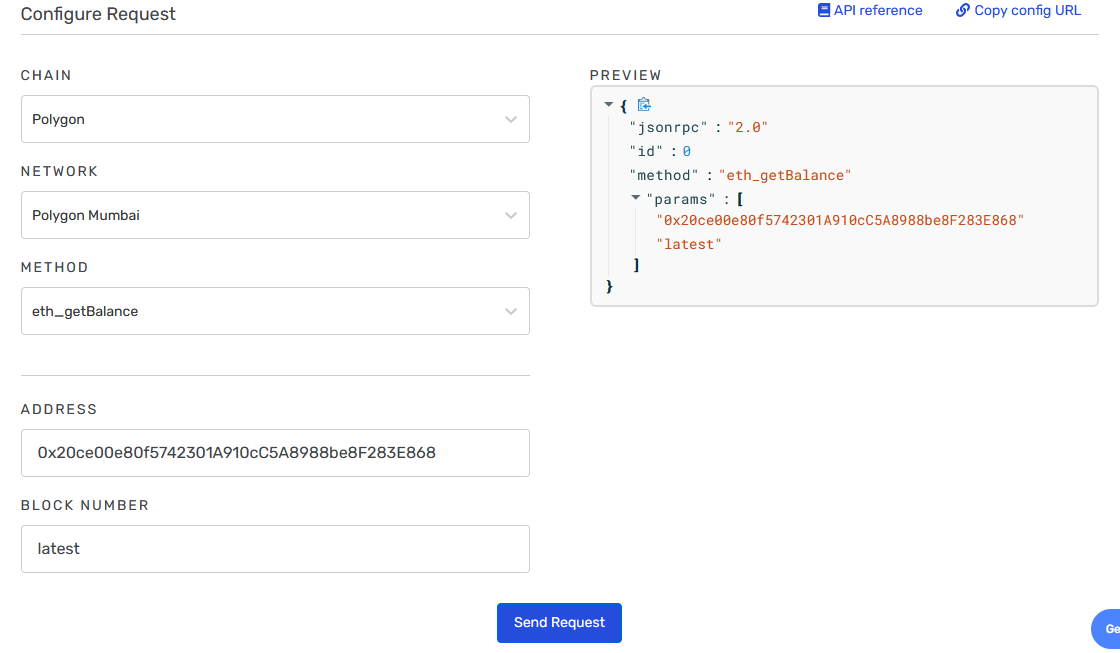


Figure 5 Request for getting the Balance of the Polygon wallet

1. Initialize the Project:

A picture containing graphical user interface

Description automatically generated

Figure 6 Creating a new folder for the project



Figure 7 Initializing the project using npm

1. Download HardHat and create HardHat project:



Figure 8 Run this insider the hello-world project

Text

Description automatically generated

Figure 9 Creating a new Hardhat project

1. 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.

1. 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.

Text

Description automatically generated with medium confidence

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

1. 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:

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.

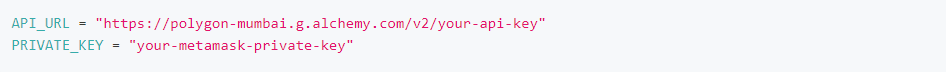


Figure 11 The .env should look like this

1. Install Ether.js



Figure 12 Command used to install ether.js

1. Update hardhat.config.js

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

Text

Description automatically generated with medium confidence

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

1. Compile our contract:

From the command line run:



Figure 14 Compiling the project

1. Write the deploy script

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

Graphical user interface, text

Description automatically generated

Figure 15 deploy.js

1. Deploy the contract:



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

1. Verify the contract

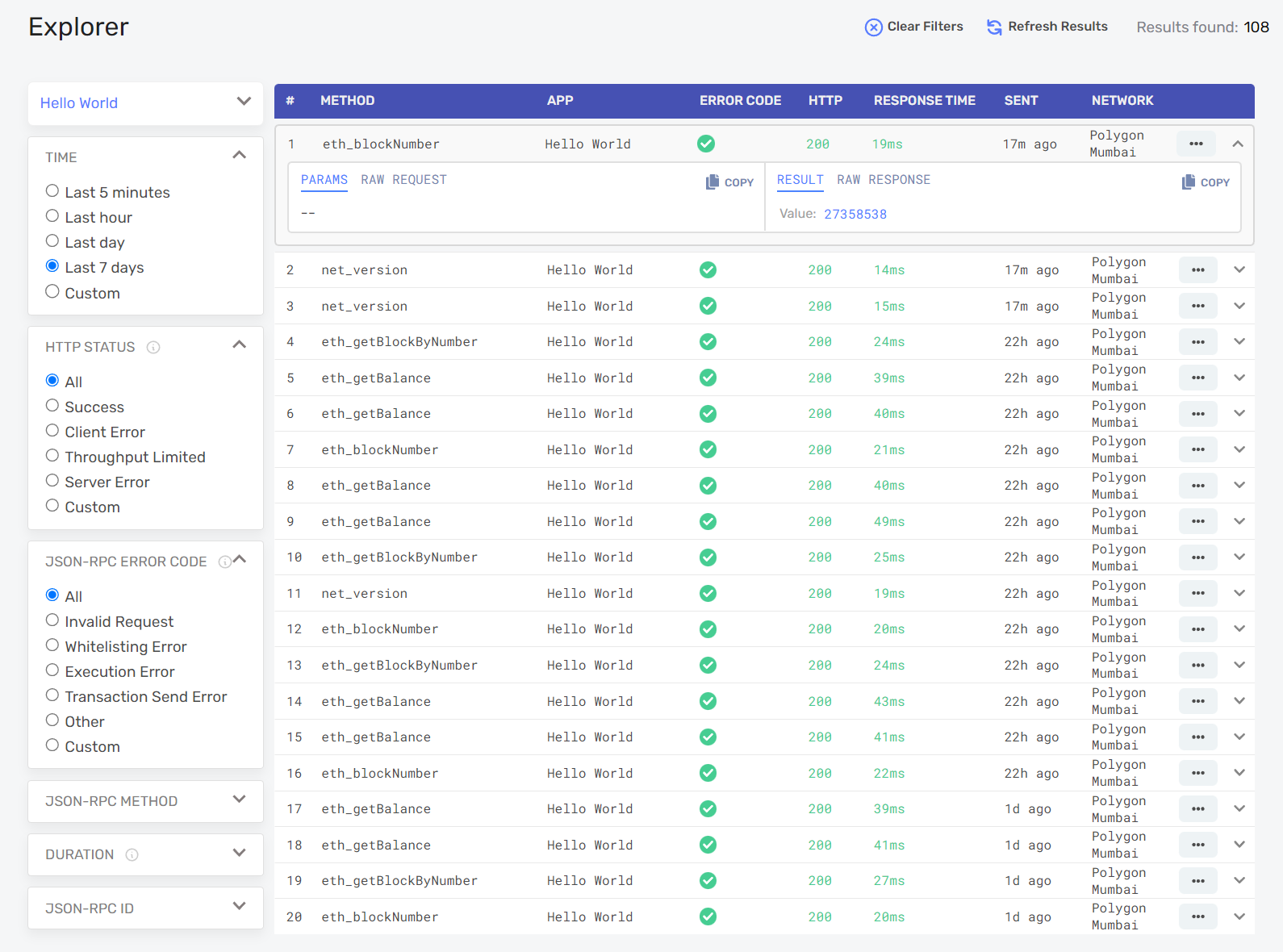


Figure 17 Using Alchemy explorer to find information about the methods deployed along the smart contract.