Compile and Deploy Using Remix IDE

Vince Millora Follow
Feb 12 · 5 min read

Remix IDE (*Integrated Development Environment*) is a web application that can be used to *write*, *debug*, and *deploy* **Ethereum Smart Contracts**. When you want to write Smart Contracts, you can use this application.

Using Remix IDE is actually quite easy! In this tutorial, we'll learn how to use it.

What We Are Trying To Do

From here onwards, we will study each part of the Remix IDE. We will write a simple Smart Contract, compile, then deploy it using Metamask **Ropsten Test Network**. Don't worry, we will guide you as you write your Smart Contract.

Function List

1. Setting a Greeting—we should be able to **set** a greeting.
2. Displaying the Greeting—we should be able to retrieve the greeting and **display** it.

Tools

1. Smart Contract
   Solidity, Remix, Metamask

Prerequisites

Before proceeding with this tutorial, you should first read the following articles:

1. How to Use Metamask

2. Basic Solidity

Studying Remix IDE

Work Flow

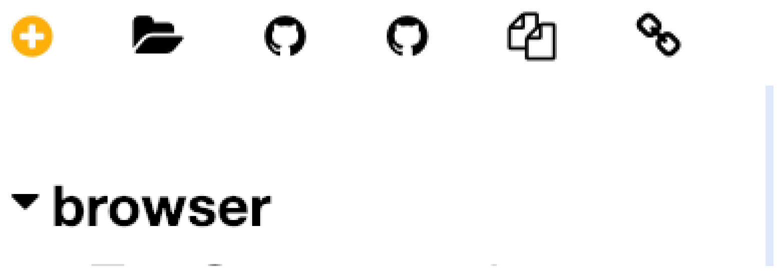
1. Creating the Smart Contract

2. Compiling and Deploying the Smart Contract

Creating the Smart Contract

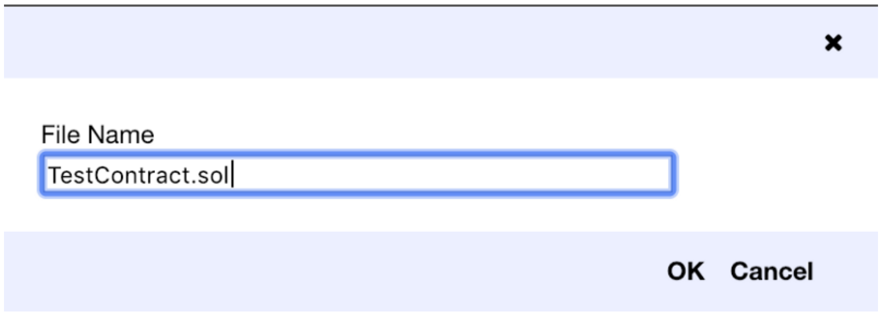
Since we really don't need a complex Smart Contract, we'll just make one for setting a greeting and displaying it. Our Smart Contract will be made using Solidity.

In Remix, create a new file named **TestContract.sol**. To create a file, click the **Create New File** button at the top left corner of the page:



Create a new file using the Create New File button.

After clicking the button, a window will appear. This will ask the filename of the new contract. Just input *TestContract.sol* like this:


File Name

TestContract.sol

After the file is created, add the following code to the code editor:

```
// We will be using Solidity version 0.5.3
pragma solidity 0.5.3;

contract TestContract {
    // Container of the greeting
    string private greeting;

    // Initialize the greeting to Hello!!.
    constructor() public {
        greeting = "Hello!!";
    }

    /** @dev Function to set a new greeting.
      * @param newGreeting The new greeting message.
      */
    function setGreeting(string memory newGreeting) pu
        greeting = newGreeting;
```

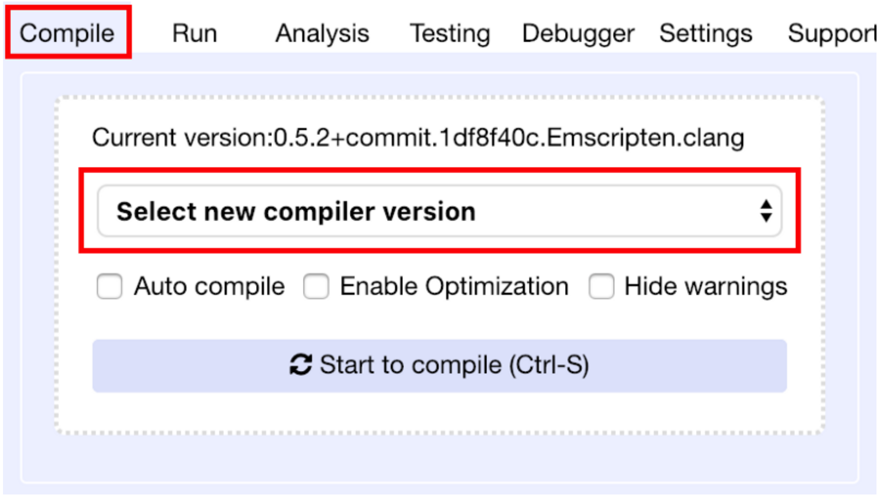
We've just basically defined two functions, *setGreeting(), greet()*, for setting a greeting and displaying it.

Compiling and Deploying Contracts

After writing the Smart Contract, we should compile the code to check if there are **errors** or **warnings**. If there are no any errors, the Smart Contract is ready to be deployed.

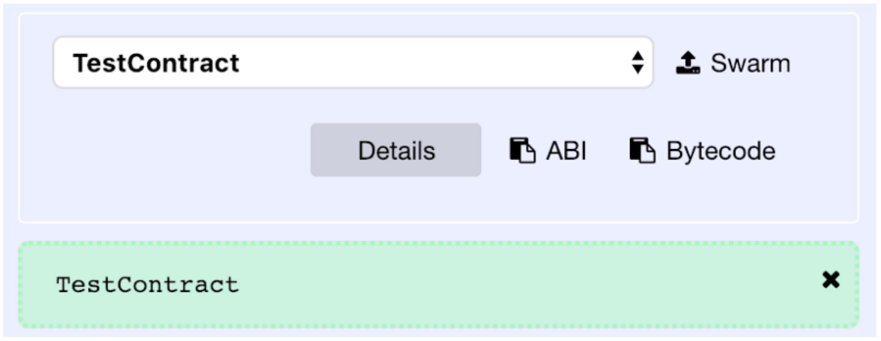
Compiling a Contract

To compile a Smart Contract, go to the *Compile Tab* and select the **compiler version** and select the appropriate version that is suitable to your **Solidity version** (in our tutorial, we will use *0.5.3+commit.10d17f24* since we used *Solidity 0.5.3)*. You can refer to the screenshot below to find the tab and the settings:



Current version:0.5.2+commit.1df8f40c.Emscripten.clang

After settings the options, click the *Start to compile* button, a success **message** or a list of **errors** will show below:

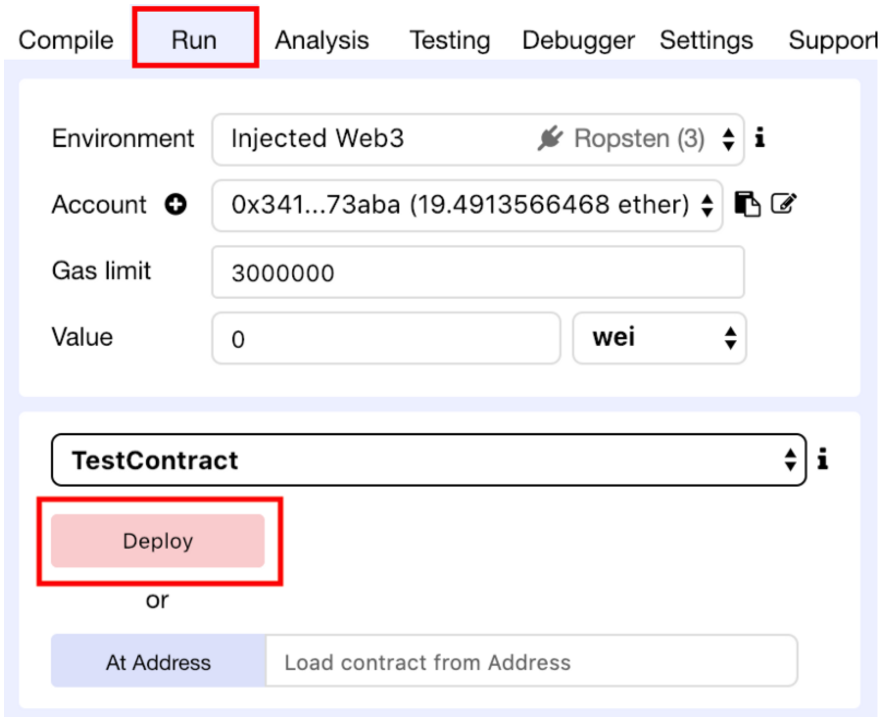


TestContract

If there are no errors, a box with a green color, then the Smart Contract is successfully compiled. You can now deploy the code or get the **ABI**. The ABI is basically a specification in the form of a JSON, that describes the contract. This is very important when you want to deploy your contract to your applications.

Deploying a Contract

After successfully compiling your contract, you can now deploy it. Deploying a contract will let you test it and use it to your applications. Go to the *Run Tab* located at the side of the *Compile Tab*:

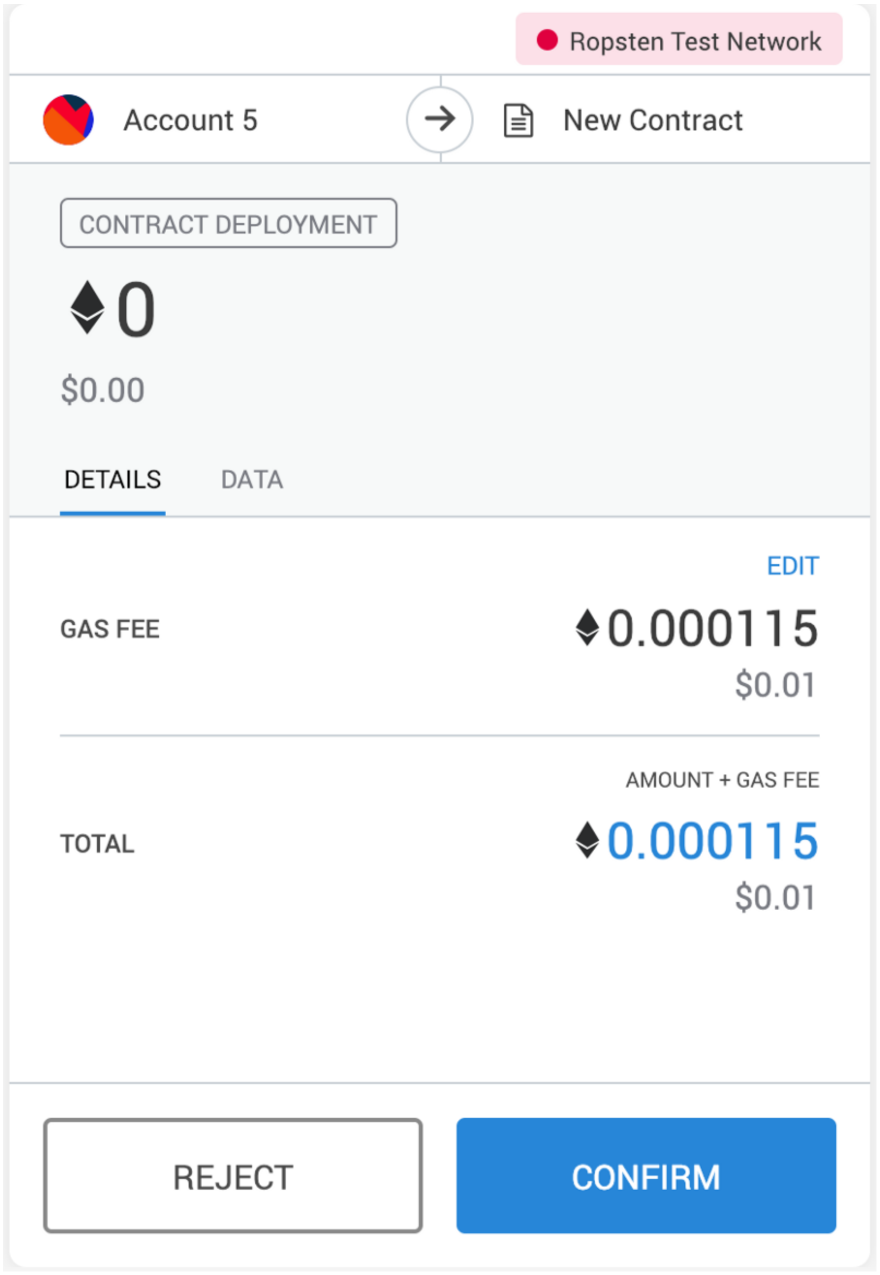


Environment: Injected Web3 — Ropsten (3)
Account: 0x341...73aba (19.4913566468 ether)
Gas limit: 3000000
Value: 0 wei
TestContract
Load contract from Address

Set the **options** first at the top part of the tab. Here are what you need to know about the options before you run a contract:

- `JavaScript VM` - The instance will be deployed in a *sandbox blockchain* in the browser. This means nothing will be saved when the page reloads, a new instance will be created.
- `Injected Provider` - Remix will connect to an *injected web3 provider*. `Metamask` and `Mist` are example of injected web3 providers.
- `Web3 Provider` - Remix will connect to a *remote instance*. You will need to provide the URL address of your selected provider.
- `Account` - the list of accounts that will be associated with the current instance.
- `Gas Limit` - the maximum gas amount of each transaction.
- `Value` - the amount of value for the next created transaction.

Since we will use the Metamask to deploy our contract, we will select the `Injected Provider` for the *Environment* and leave the other options to default. Then click the *Deploy* button to deploy the contract. A Metamask window will appear and you will need to confirm to deploy the contract:


Ropsten Test Network

Account 5 → New Contract

CONTRACT DEPLOYMENT

♦0

$0.00

DETAILS DATA

EDIT

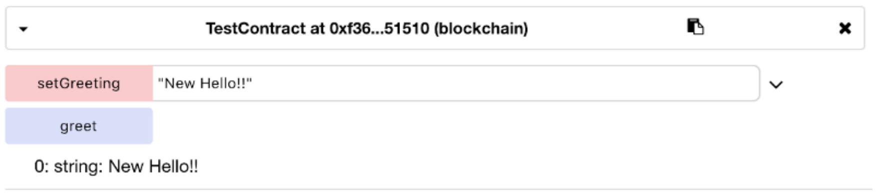
GAS FEE ♦0.000115 $0.01

AMOUNT + GAS FEE

TOTAL ♦0.000115 $0.01

REJECT CONFIRM

After the contract is deployed, the *instance* will show up at the bottom of the tab and you will be able to access the functions. In this example, we set the greeting to *New Hello!!* and then displayed it.



"New Hello!!"

0: string: New Hello!!

After testing, you can also copy the Smart Contract's **instance address** using the *copy* button beside the address. This address basically tells where your contract is running. Along with the ABI, these two are the required contract information that will be used to connect the contract to your applications.

And we're done. We've created a Smart Contract, compiled it, then deployed it.

Conclusion

Now, you know how to use the Remix IDE. You can now write Smart Contracts in Remix, compile it, and deploy.

Remix is very helpful when you are just starting to learn programming in Ethereum. Every basic thing you need to deploy your Smart Contract can be found in the IDE.

So what's next?

You can now study more about Ethereum and practice coding using the Remix IDE. You can also study Web3.js, a tool to connect the **Smart Contract Instance** to your **Web application**. OR, you can practice developing Dapps by viewing openberry's other tutorials.

openberry is a tutorial marketplace, designed to allow anyone to learn blockchain programming.

openberry | blockchain tutorial

Anyone using openberry can become a blockchain engineer. Openberry is a blockchain tutorial…

www.openberry.ac

openberry (@openberry_ac) | Twitter

The latest Tweets from openberry (@openberry_ac): "https://t.co/o1leXC5rUL"

twitter.com