Ethereum Studio Docs

Obsidian Labs

Feb 14, 2022

GETTING STARTED

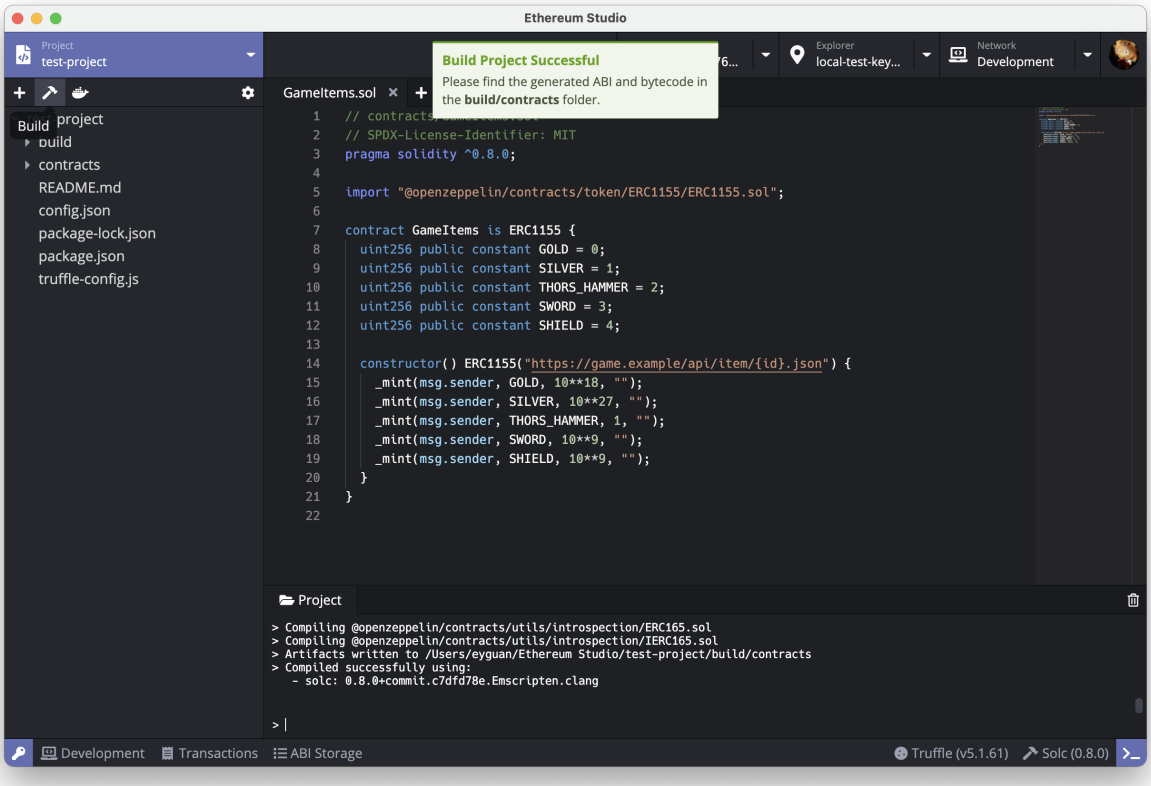
CHAPTER

ONE

OVERVIEW

The Ethereum Studio is a world-class Ethereum smart contract and DApp integrated development environment (IDE), aiming to make Ethereum development faster and easier. Ethereum Studio currently offers a standalone desktop application running on macOS, Windows and Linux, and Ethereum Studio Web that runs in modern web browsers. With Ethereum Studio, you can:

- Set and save a project in the cloud or local quickly
- Manage keypair information with MetaMask easily
- Build and test smart contracts with different framework
- Deploy smart contracts to Ethereum mainnet and testnet
- Check and call deployed contract functions through the address
- Query address information with Explorer on the selected network
- Setting local RPC node, ABI Storage and other advanced features



display

When deploying contracts, you should use the latest released Docker image version. Apart from exceptional cases, only the latest version receives security fixes. Furthermore, breaking changes as well as new features are introduced regularly. We currently use a 0.y.z version number to indicate this fast pace of change.

Ideas for improving Ethereum Studio or this documentation are always welcome. Read our **contributors guide** for more details.

CHAPTER

TWO

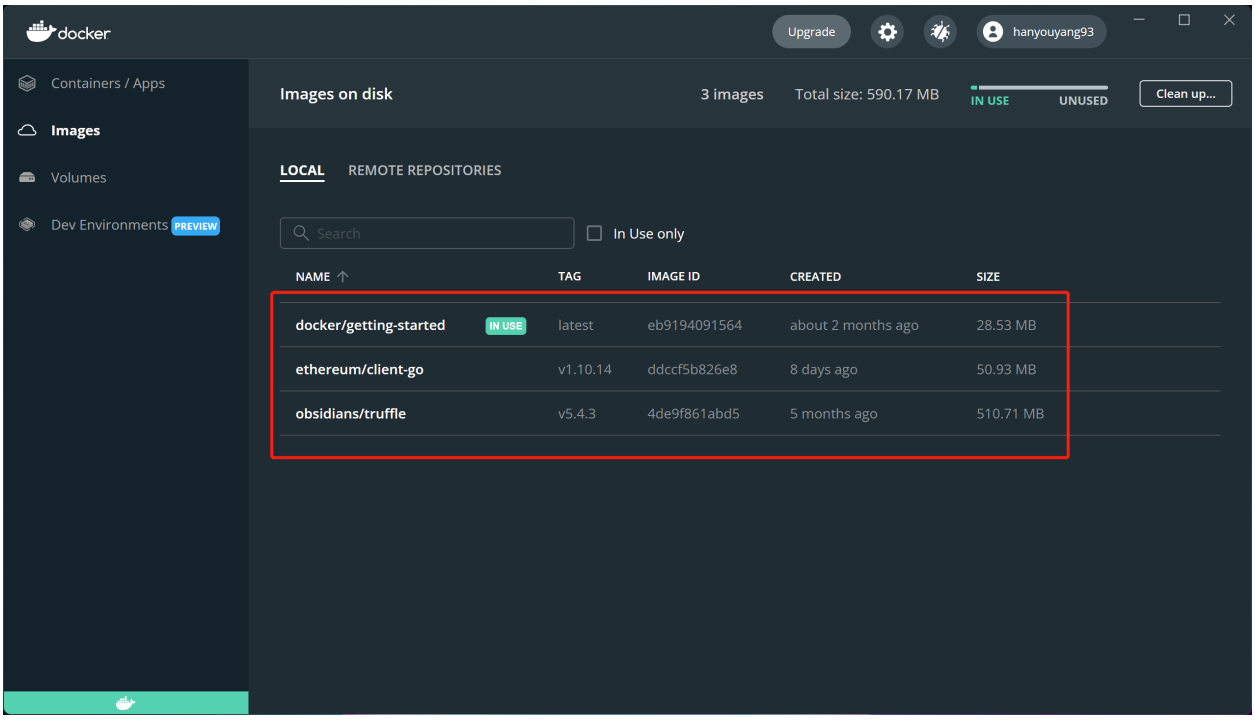
INSTALLATION

2.1 Download

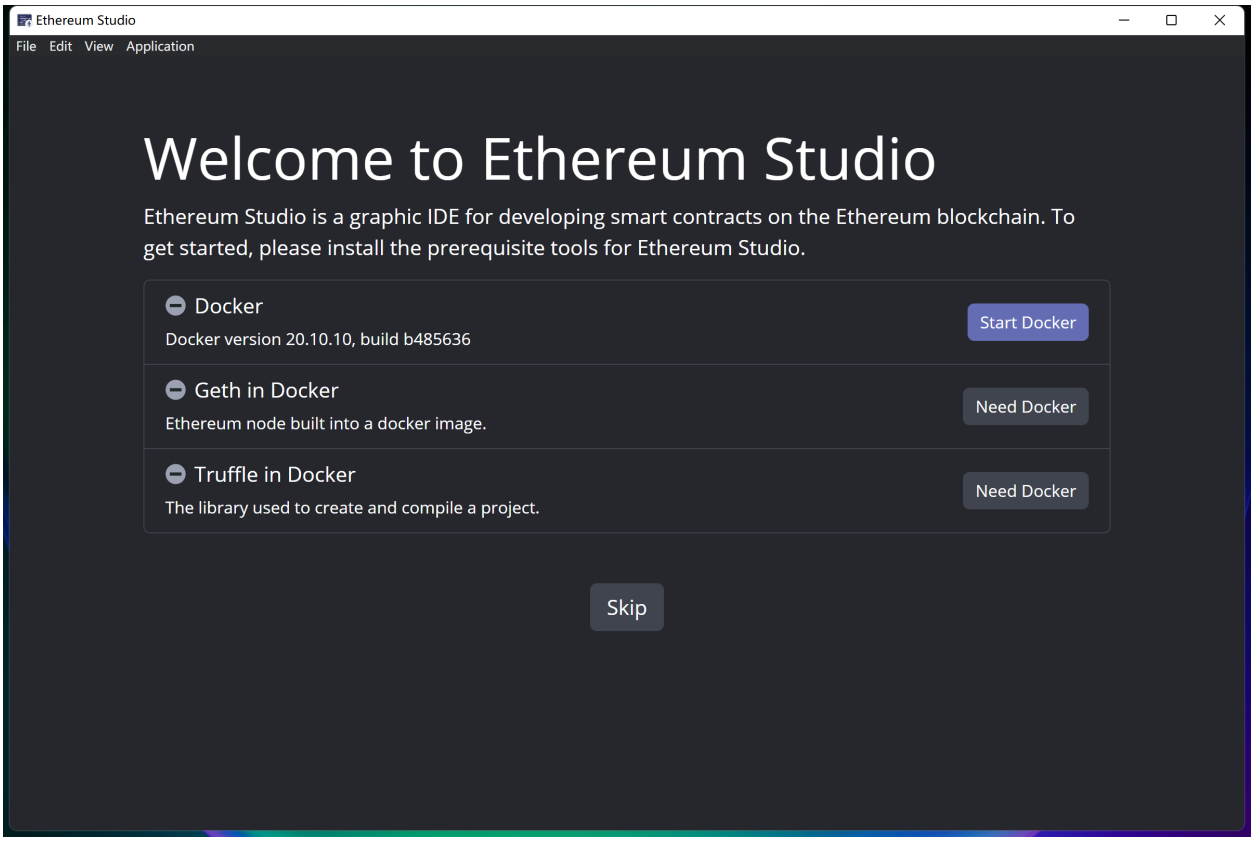
Download Ethereum Studio installation package in Github Ethereum Studio Latest Release according to the computer system type (`.dmg` for **macOS**, `.AppImage` for **Linux**, `.exe` for **Windows**).

2.2 Install Dependencies

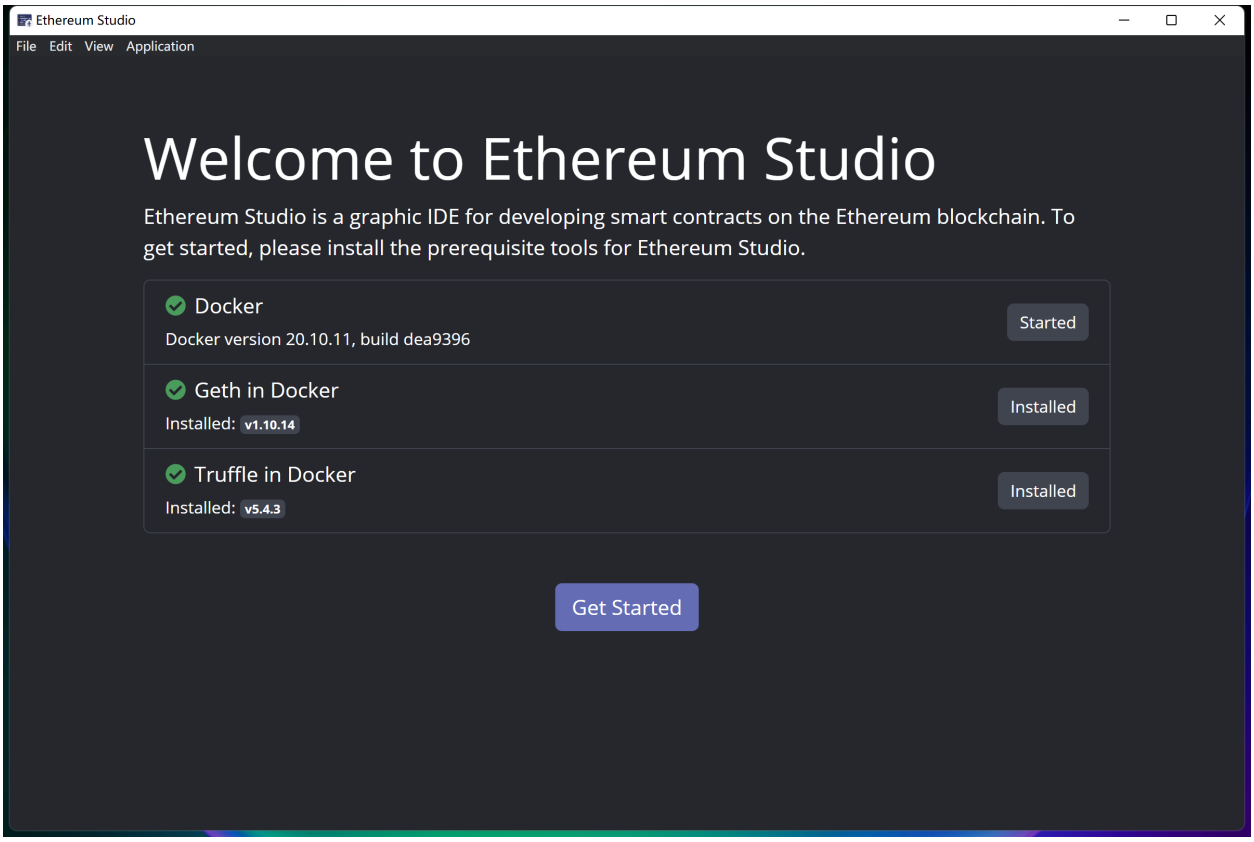
Starting Ethereum Studio requires several image dependencies, including Docker, Ethereum Node and Ethereum Truffle. Running Ethereum Studio desktop client requires all three image files. The modules are installed in Docker client like this:



Install image files through the “Welcome” page, if there is any missing dependency in Docker. Select the available versions in the “Geth in Docker” list. It is the same for “Truffle in Docker”, too.



Click the “Get Started” button, and “Project” interface will pop up, after all the required dependencies are installed.



2.2.1 Docker

Docker is used to start the Ethereum Node and build projects in Ethereum Studio. If Docker is not installed yet, users can click the "Install Docker" button to visit the official Docker website and download and install it.

Docker can not wake up automatically through the Ethereum Studio desktop client now. Open Docker desktop client before starting the the Ethereum Studio desktop client. Otherwise, an error report would remind "Docker has not been installed".

2.2.2 Geth

Geth in Docker is the Ethereum node image. Ethereum Studio uses the image to run the Ethereum node and build projects. Install the Geth image through the "Install" button and select the required version. The latest version is always recommended as the beginner's default version.

2.2.3 Truffle

Truffle in Docker is an Ethereum version of Truffle used to create and build projects. Install the Truffle image through the "Install" button and select the required version. The latest version is always recommended as the beginner's default version.

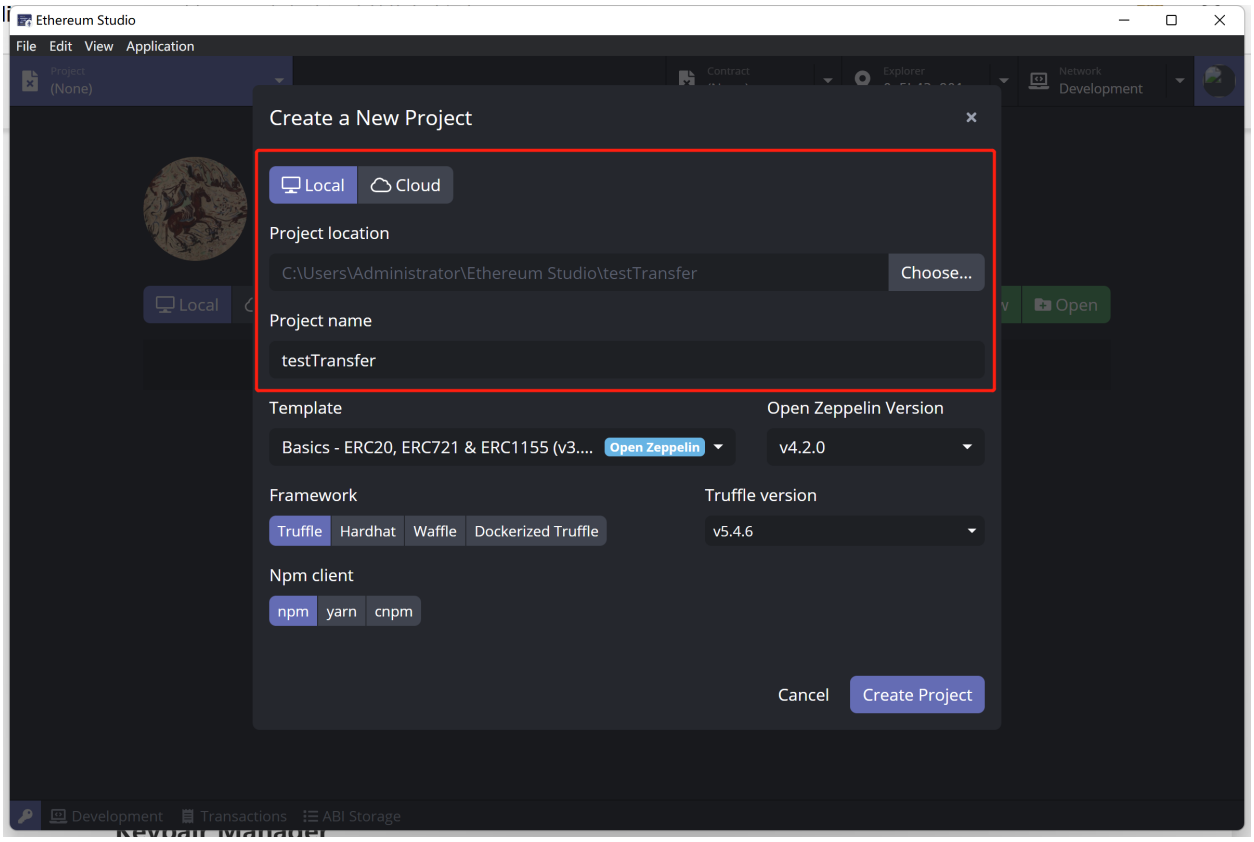
The grey "Skip" button will change into a violet "Get Started" button after all dependencies are correctly installed. Click it to enter the main interface of Ethereum Studio.

2.3 Contrast with Web Client

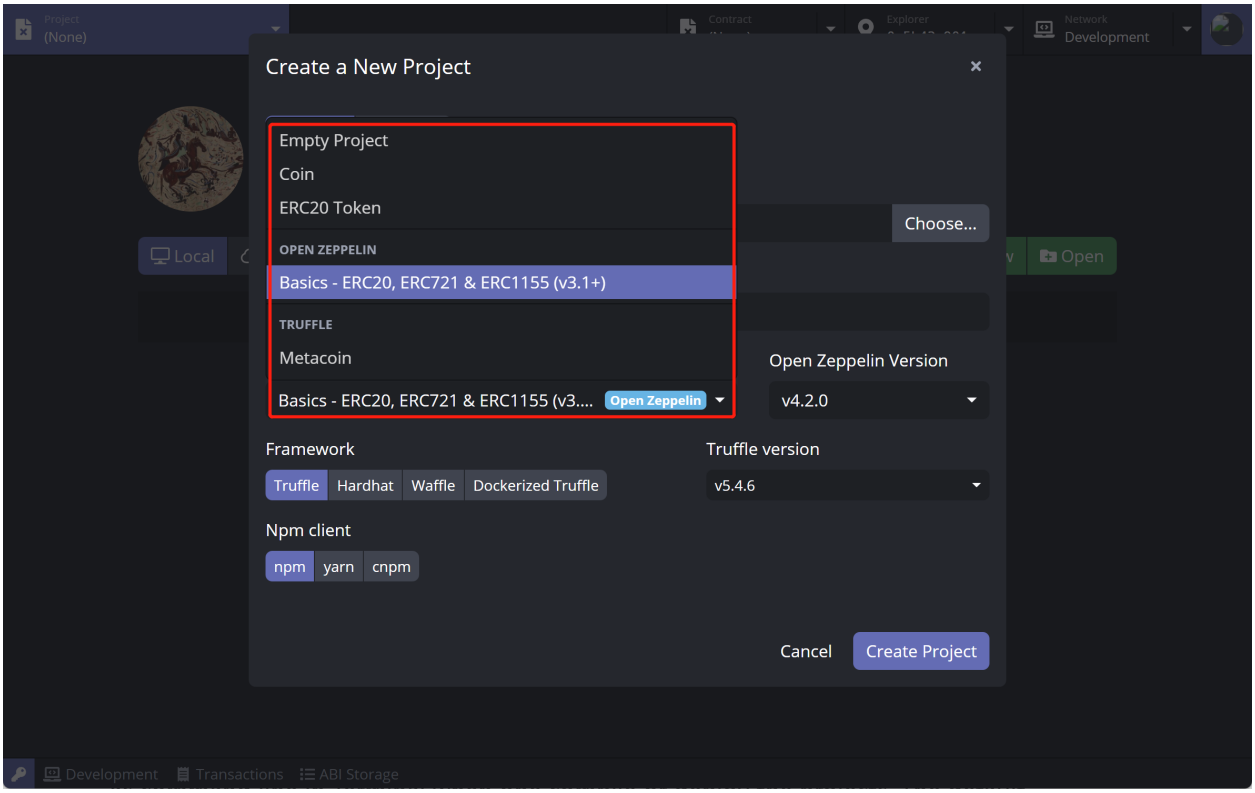
Features		Web	Desktop
Projects	Cloud	Yes	Yes
	Local	No	Yes
Framework Choices	Truffle	No	Yes
	Hardhat	No	Yes
	Waffle	No	Yes
Build	by Framework	Only Truffle	Truffle, Hardhat, Waffle and Dockerized Truffle
	by Command Line	No	Yes
Deploy	by Framework	Only Truffle	Truffle, Hardhat, Waffle and Dockerized Truffle
	by Command Line	No	Yes
Login Account by Github		Yes	Yes
Auto-Log Address by MetaMask		Yes	No
Blockc Explorer		Yes	Yes
Contract		Yes	Yes
Network	Developemt	No	Yes
	Testnets and Mainnet	Yes	Yes
	Custom	No	Yes

2.3.1 Create a New Project

Creating an ERC20 project in the Ethereum Studio Desktop client is a bit different, compared with creating it in web client. The "Project name" is "testTransfer" and the "Project location" is automatically settled as "C:\Users\Administrator\Ethereum Studio\testTransfer" in the Windows system. The "Project location" must be an empty documentation. The web client will save projects in the cloud automatically.



In the Ethereum Studio desktop client, set "Template" as "Basics - ERC20, ERC721 & ERC 1155" or Truffle framework "Metacoin" to save time importing basic projects in the later development process. While in the web client, there are only three types to choose from – "Empty Project", "Coin", and "ERC20 Token". The "Open Zeppelin Version" is the latest version automatically.

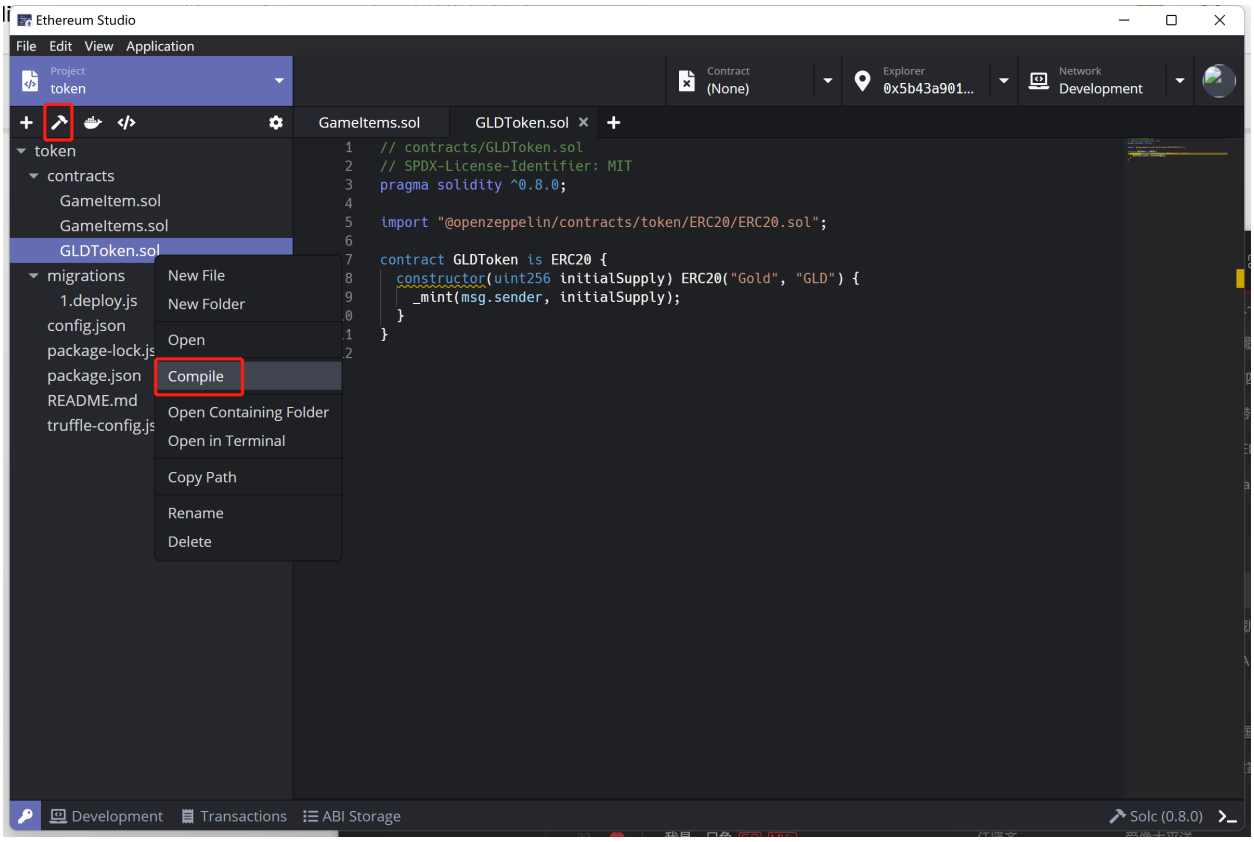


Select one of the three different frameworks as the development environment in the Ethereum Studio desktop client. The frameworks are “Truffle”, “Hardhat”, “Waffle”, and “Dockerized Truffle”. Besides, there are also three types of “Npm client”, which are “npm”, “yarn”, and “cnpm” that will not show in the web client.

Those frameworks and tools will be used automatically in the command line when building or deploying projects. The default framework version is settled during the install process of Docker, and it will be introduced later.

2.3.2 Build contracts

In the Ethereum Studio desktop client, build a Solidity file through “Right Click”, the file name. In contrast, in the web client, clicking the “hammer” icon is the only way to build.



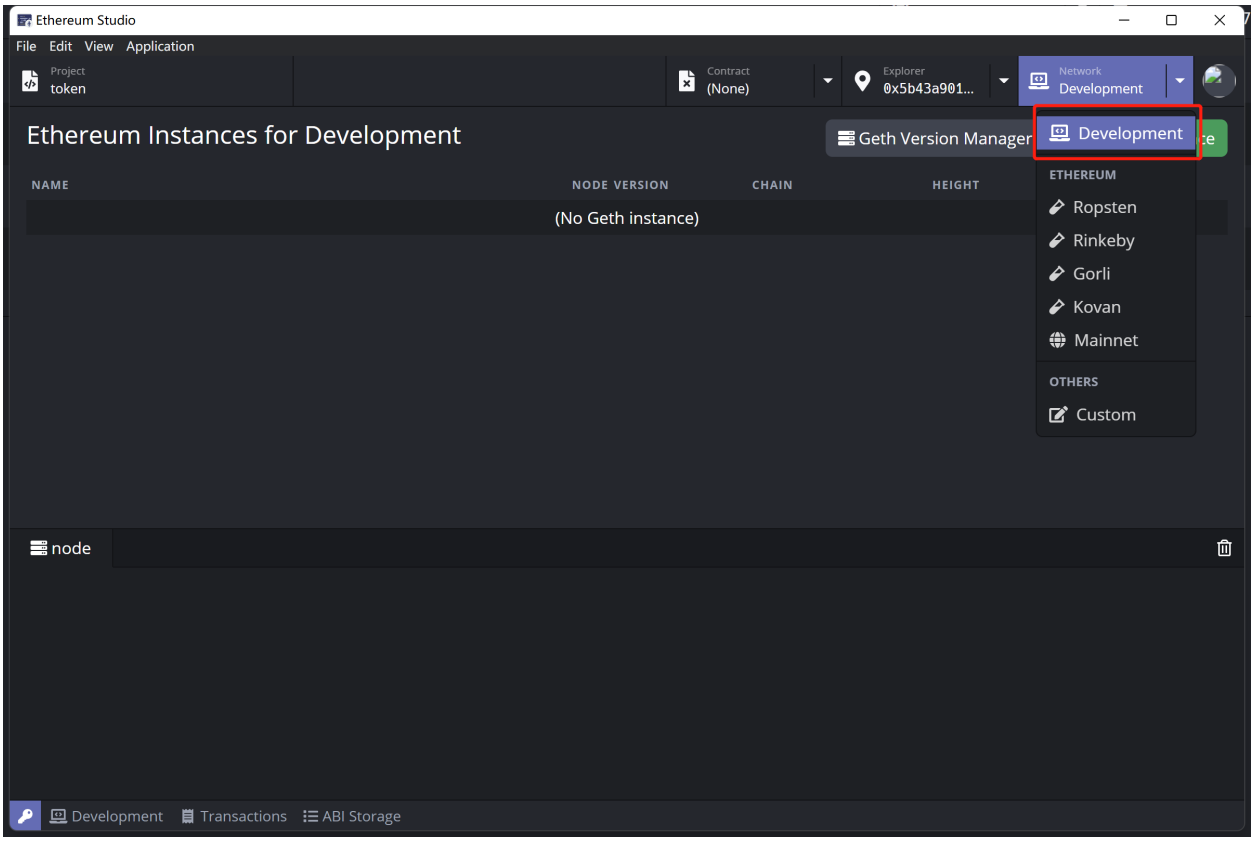
2.3.3 Keypair Manager with MetaMask

The Ethereum Studio web client will wake up the extension of MetaMask automatically in browser. After logging in to the MetaMask account, the Block Explorer will link to the account. Check the detailed information about it. Remember to import the mnemonic in the Keypair Manager to use it in the later paying gas fee in deploying.

Import a MetaMask account through mnemonic in Keypair Manager in the Ethereum Studio desktop client since the browser extension is invalid in the desktop client.

2.3.4 Development and Custom Network

Select “Network” as “Development” to set a local instance in the Ethereum Studio desktop client while the web client has no local network.


Ethereum Studio
File Edit View Application
Project
token
Contract
(None)
Explorer
0x5b43a901...
Network
Development
Ethereum Instances for Development
Geth Version Manager
Development
NAME
NODE VERSION
CHAIN
HEIGHT
(No Geth instance)
ETHEREUM
Ropsten
Rinkeby
Gorli
Kovan
Mainnet
OTHERS
Custom
node
Development
Transactions
ABI Storage

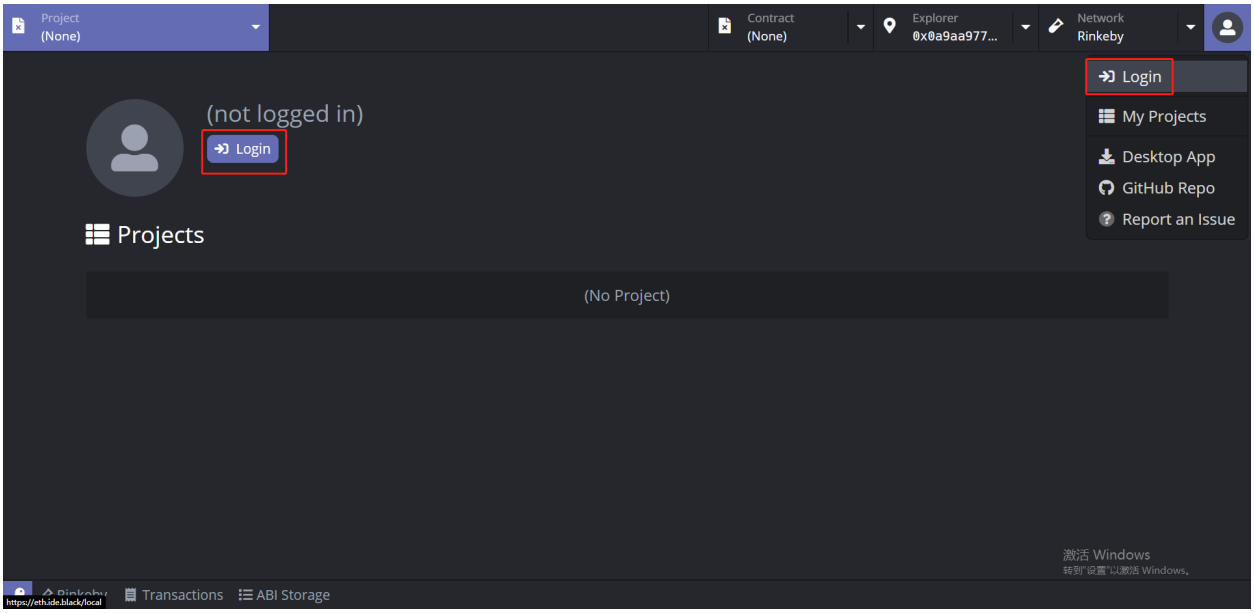
CHAPTER

THREE

QUICKSTART

3.1 Login web client

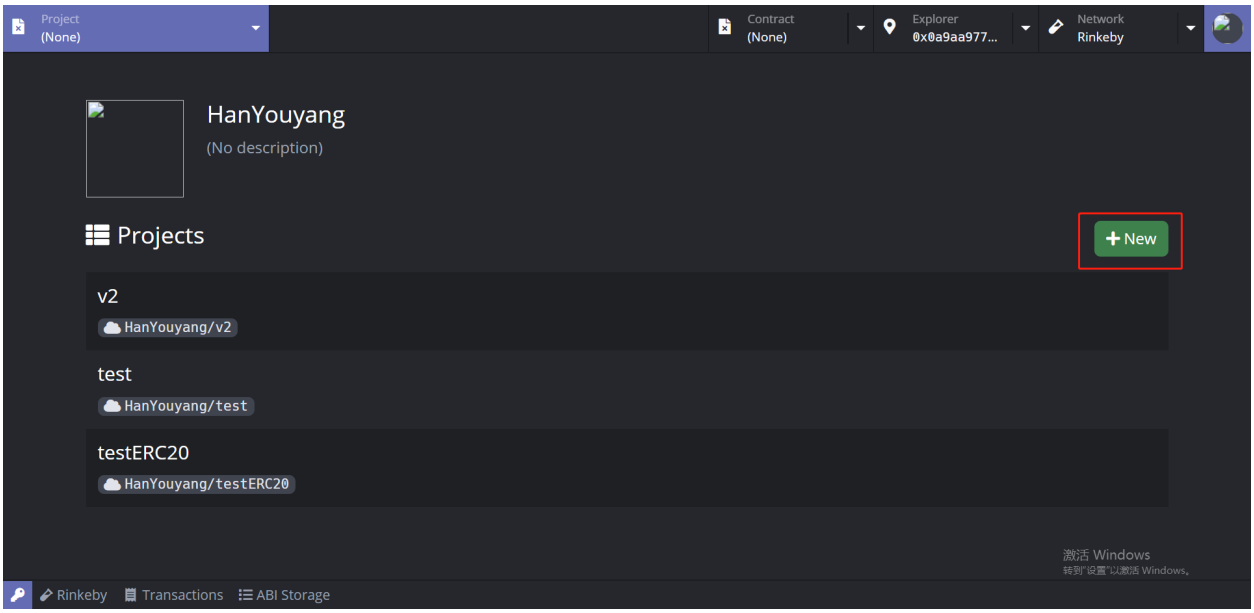
Click the “Login” icon on the bottom of “not logged in”. Otherwise, click the upper right corner and click the “Login” on the panel.



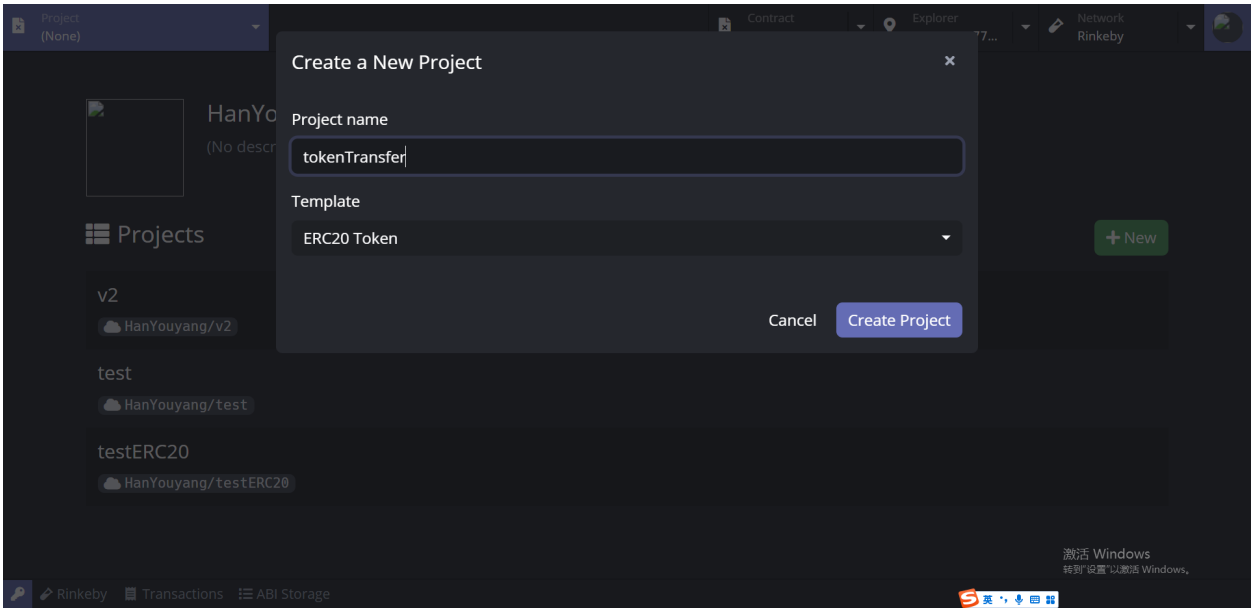
Users login through their Github accounts, in the Ethereum Studio. Once logged, users log again through Github information automatically. The Ethereum Studio will not keep users’ accounts information.

3.2 Create an ERC20 project

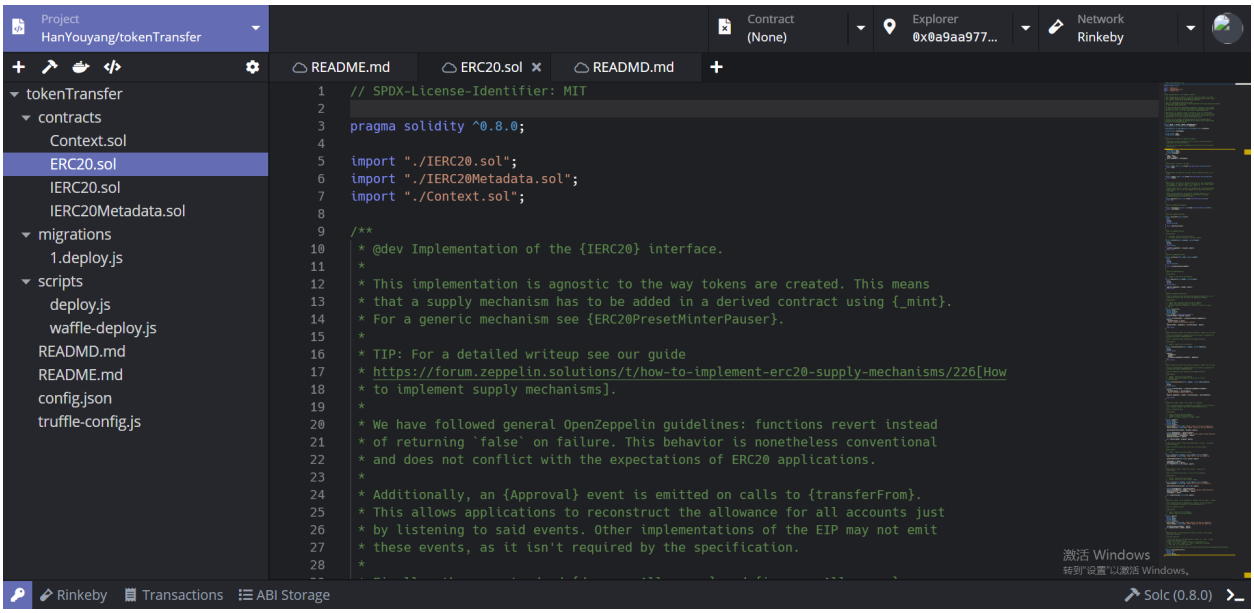
After login, create a new project by clicking the green button with “New”.



Create an ERC20 project in Ethereum Studio. Set "Project" as "tokenTransfer" and "Template" as "ERC20 Token". The project will be automatically saved in Ethereum Studio's cloud under one's account. Then click the purple button at the bottom left corner to create.

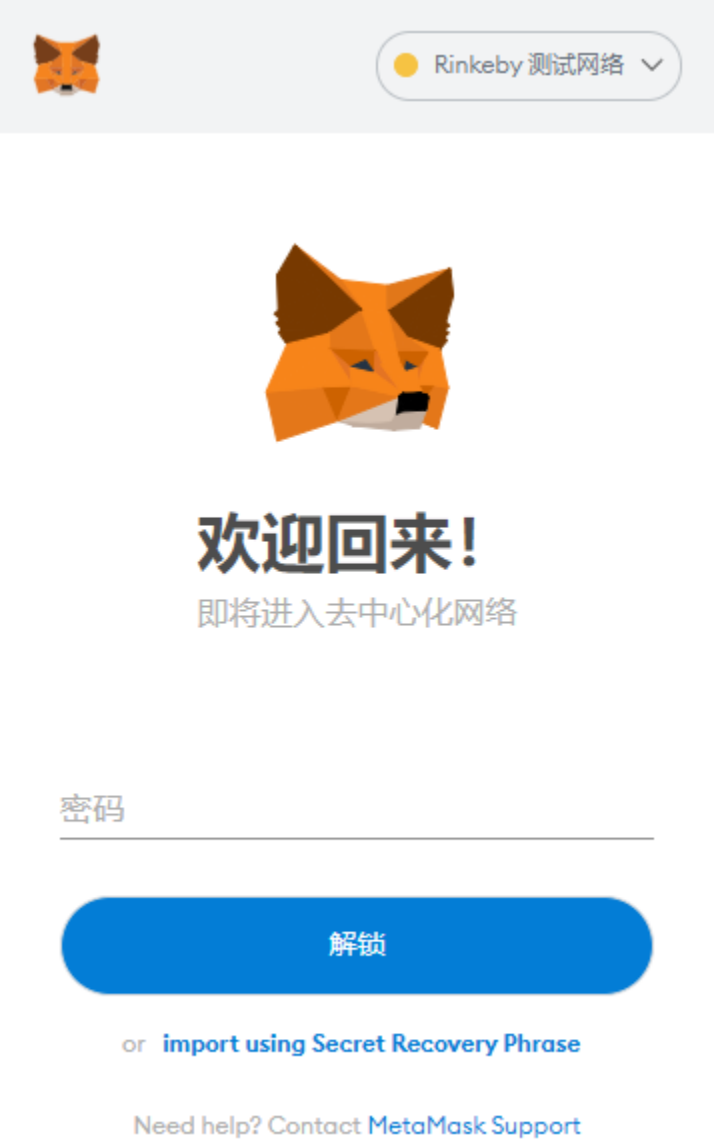


After the project is created, the ERC20 contracts have been generated successfully.



3.3 Connect to MetaMask Account

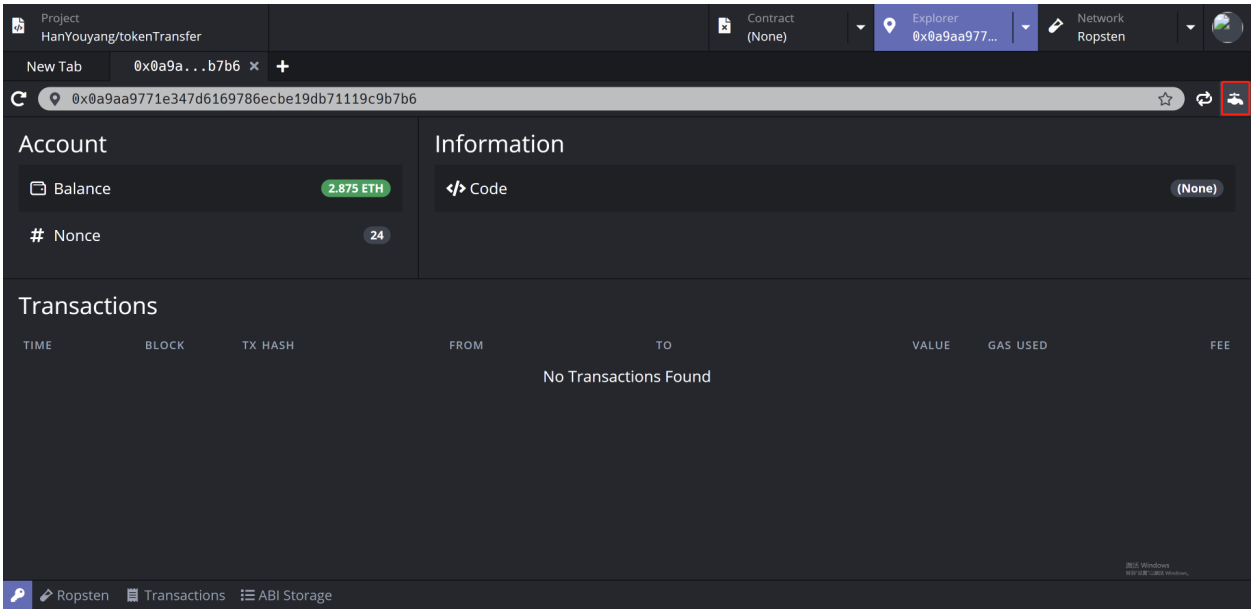
A MetaMask wallet pop up in the web client to login through users' passwords. Suppose a user does not have a MetaMask account. Please refer to the MetaMask register introduction.



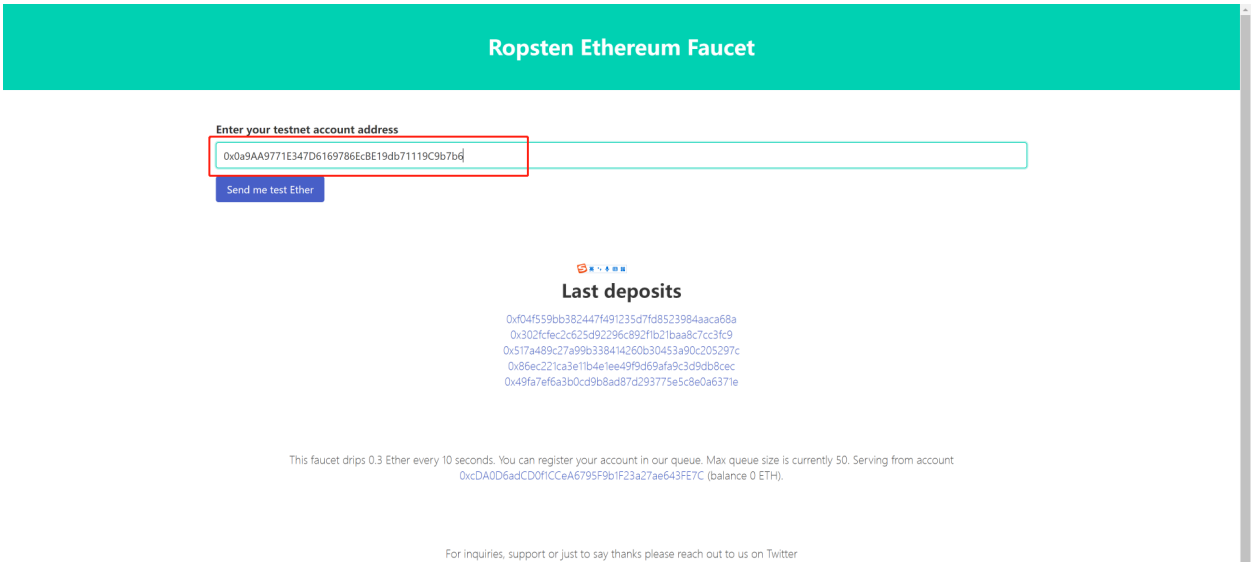
After linking to the MetaMask, copy the mnemonic into the Keypair Manager for later deploying contracts. Click the purple key icon on the bottom left corner and then click the “Import” button and paste the mnemonic words, 12 English Words representing the private key. Finally, give this imported private key a name, and the Etereum Studio will save it under this account in the cloud.

3.4 Request Ropsten Faucet

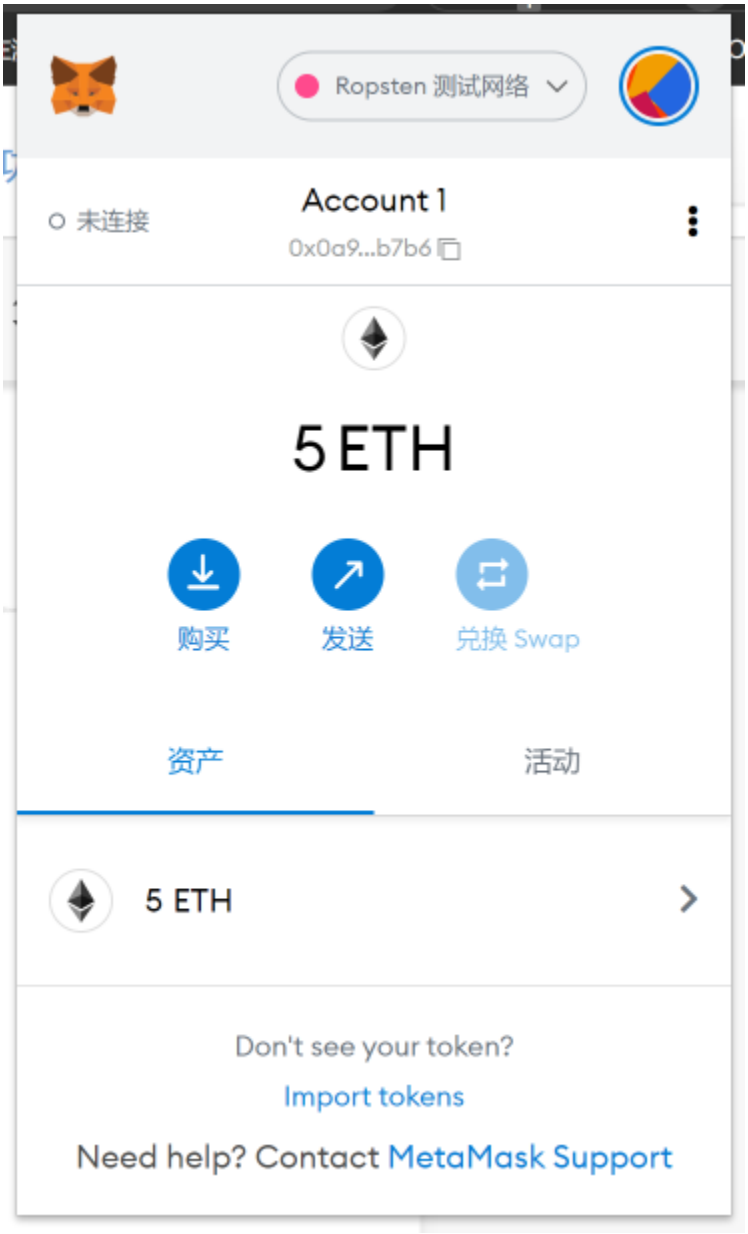
After choosing the "Network" and "Explorer", click the upper right "Faucet" icon and get the request page of testETH on the testnet. Choose "Ropsten" testnet and click the icon to turn to the request page.



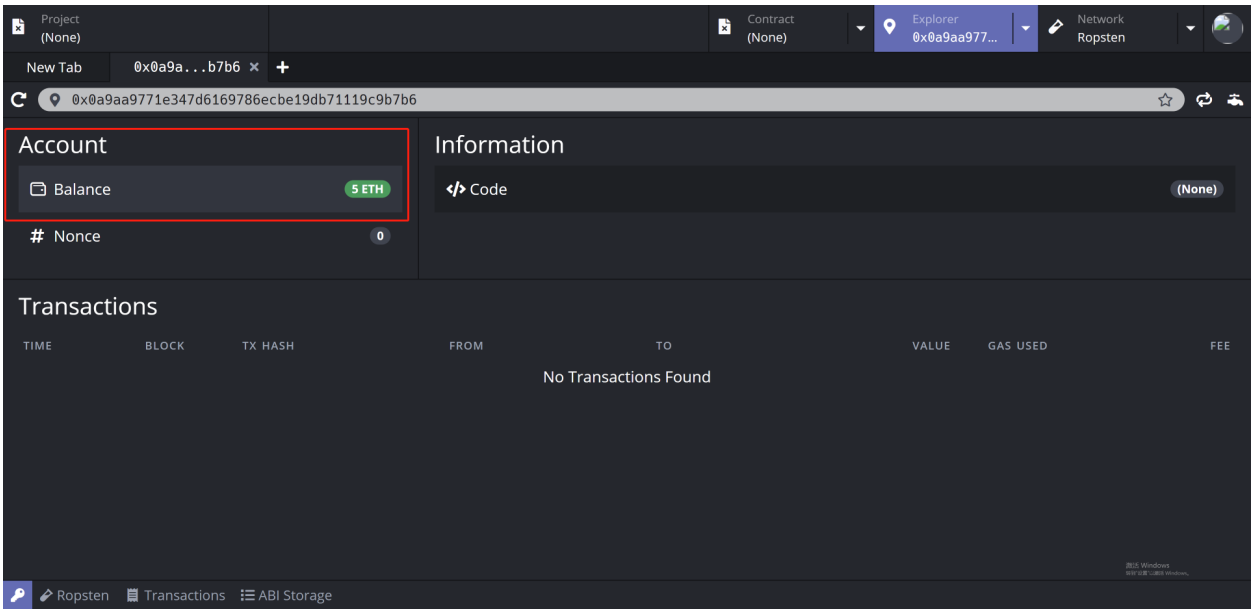
Click the upper right "Faucet" icon, and the page will jump to the Ropsten Faucet. One can copy the wallet address in MetaMask and paste it into the box. Then clicks the button "Send me test Ether" and wait for a few minutes.



With all the requests finished, there will be 5 testETH in the wallet address on Ropsten test network.

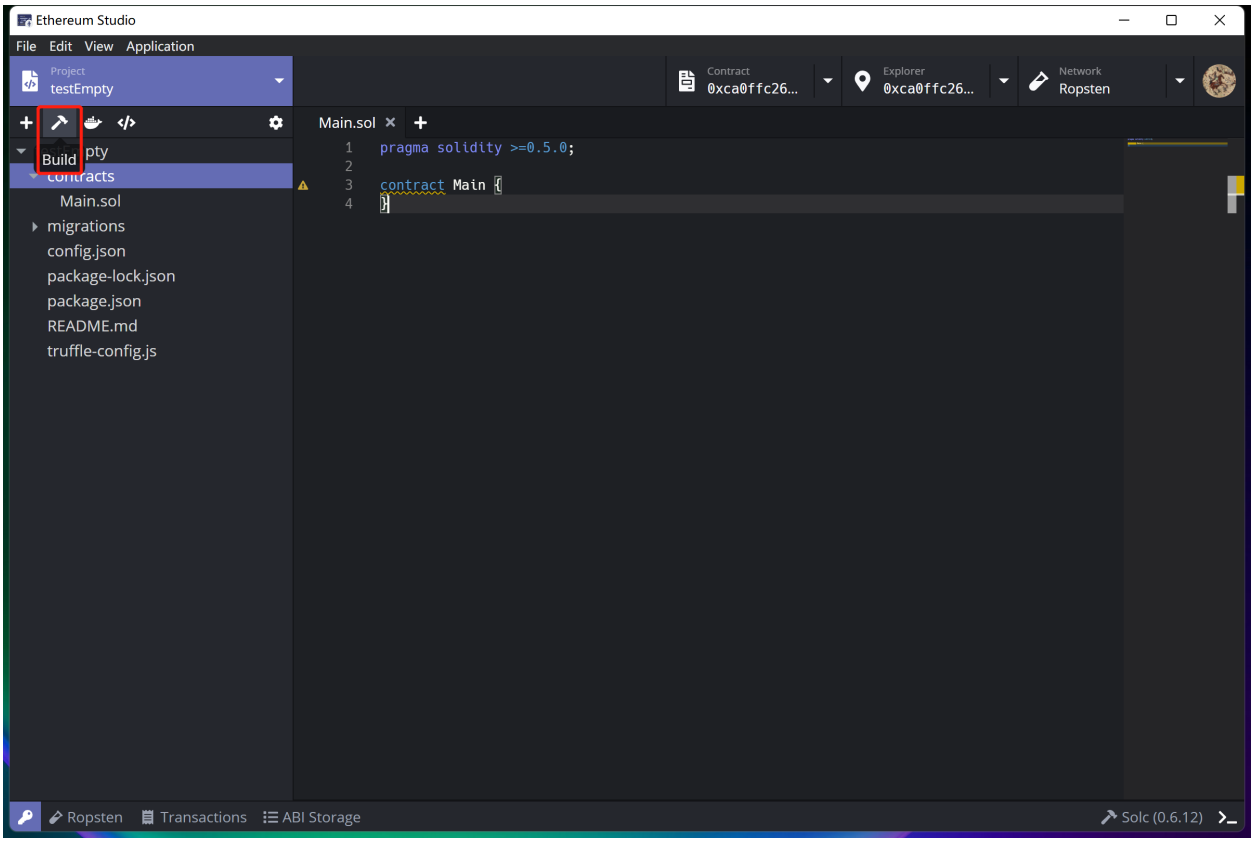


Back to the web client, check the balance if there is 5 testETH on the test network, Ropsten.

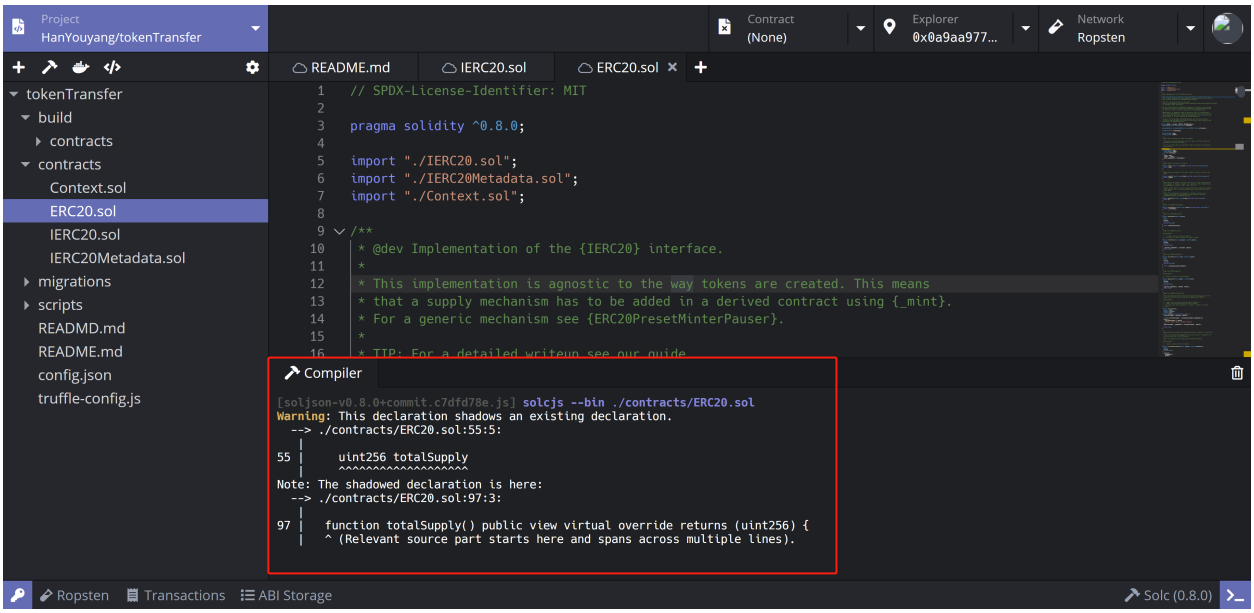


3.5 Build Contracts

There is three Solidity files in the "ERC20/contracts/" file in the left panel after successfully creating the project. Select one of the contract and click the "Hammer" icon to build the contract.

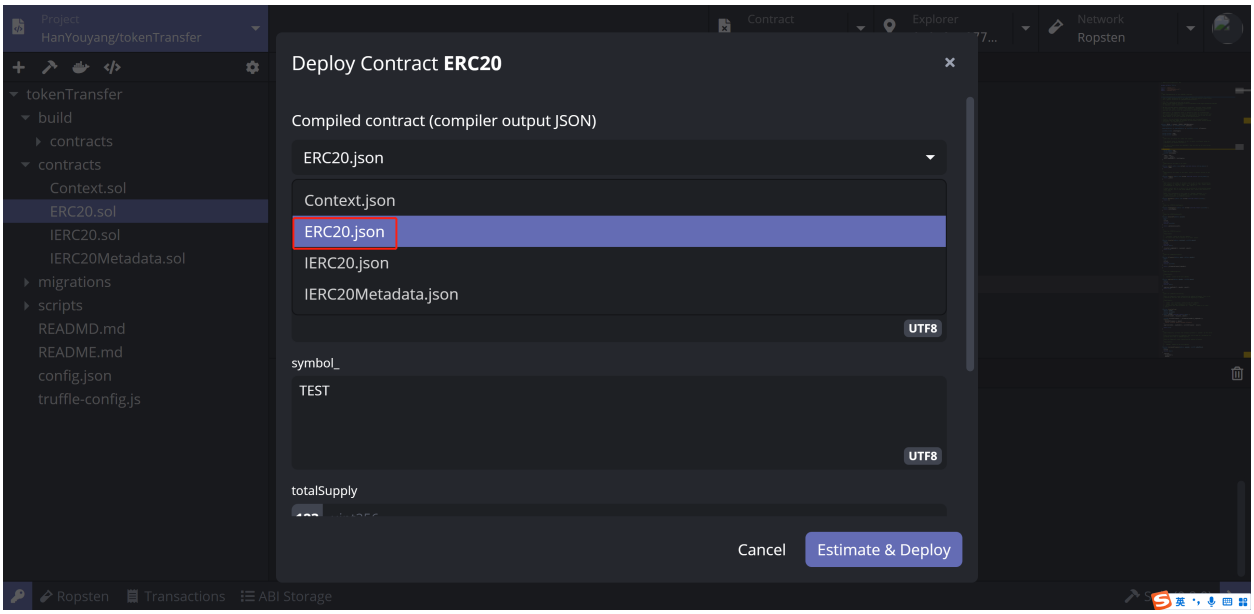


Now this IDE only supports building all contracts together. Single-file compilation will come soon in the later version.

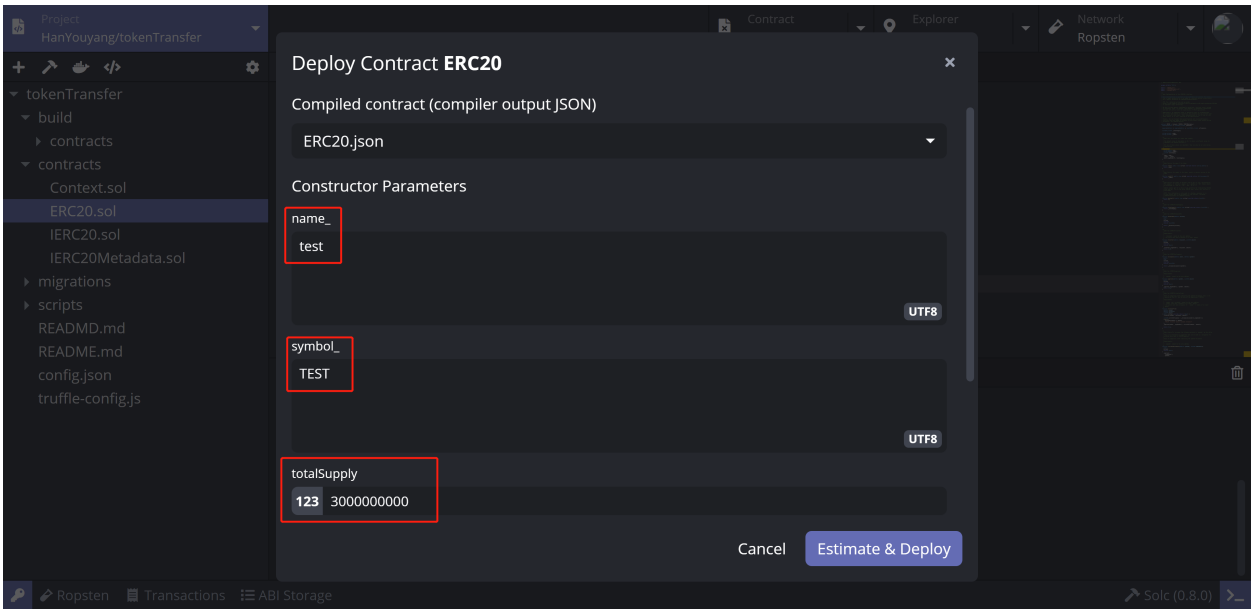


3.6 Deploy Contracts

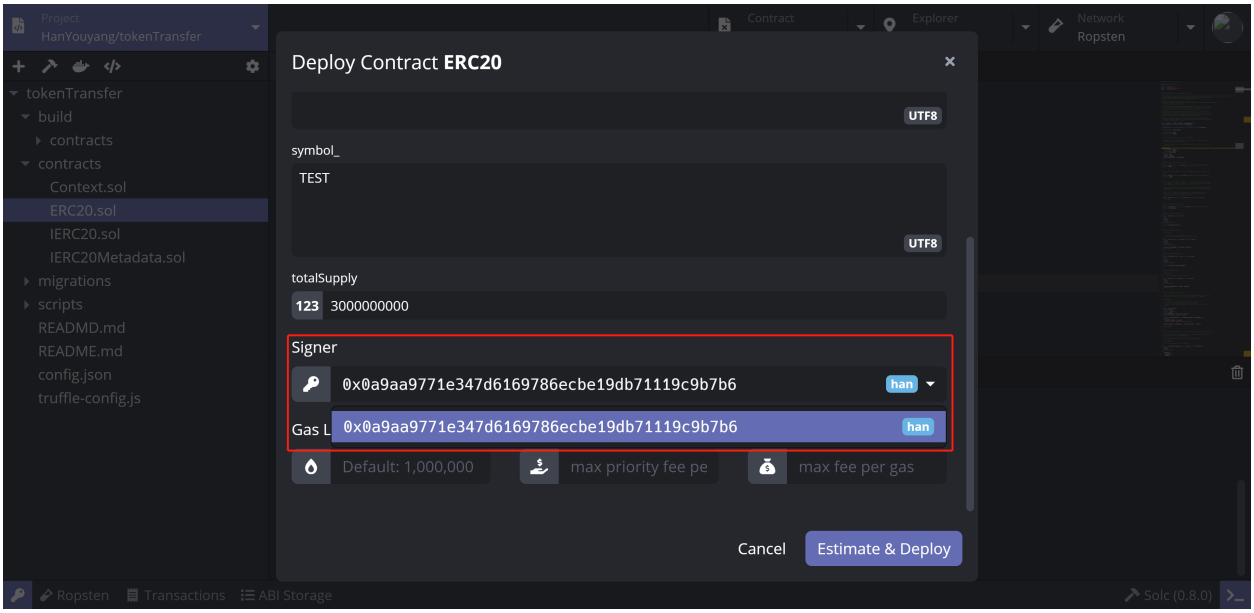
After a successful building, deploy the ERC20 contract on the Ropsten testnet. Select the JSON file “ERC20.json”.



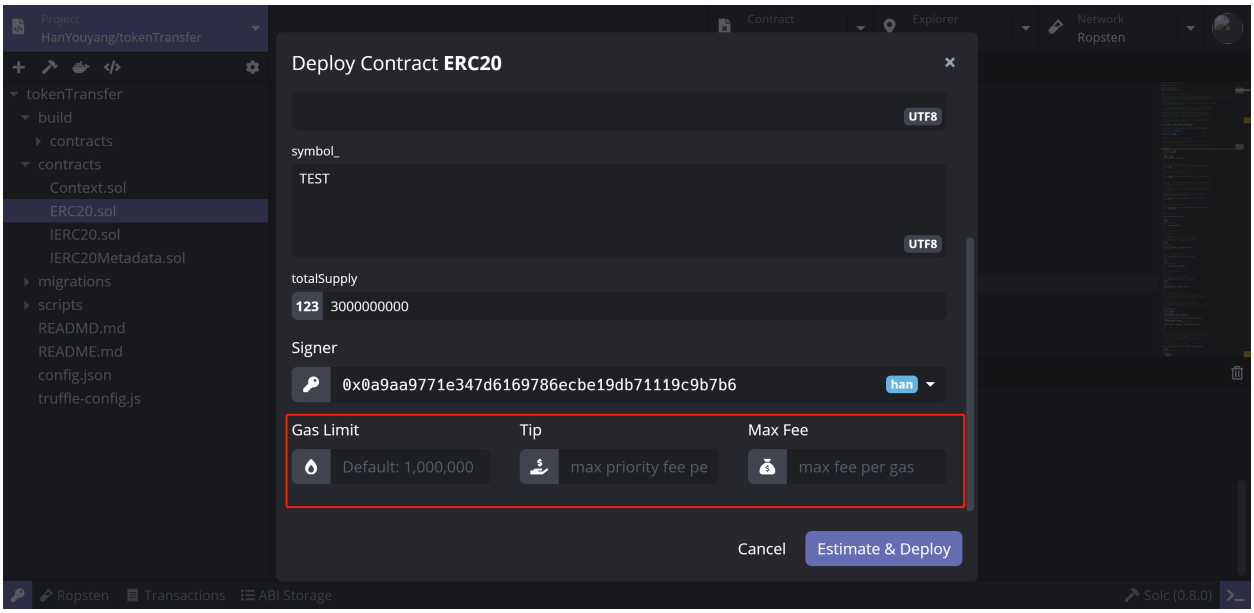
Then set “name_” as “test”, “symbol_” as “TEST” and totalSupply as “3000000000”.



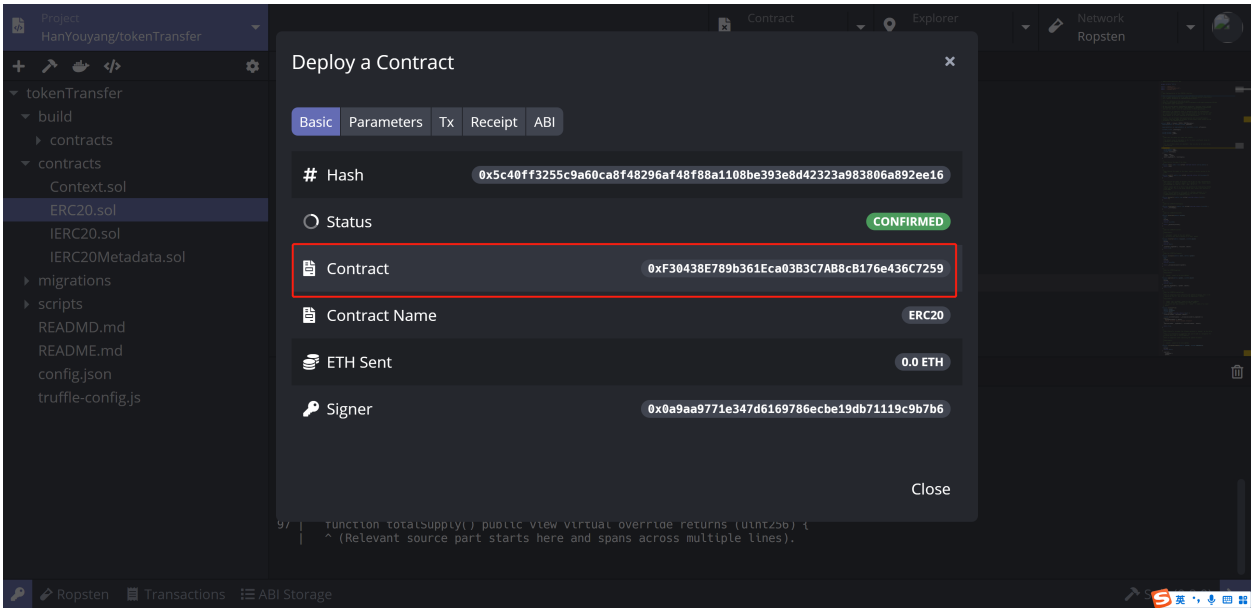
Select the signer as the keypair saved in Keypair Manager. If there is no choice, please check the chapter **Keypair Manager**.



Set the “Tip” as desired amount, or generate it with “Gas Limit” and “Max Fee” together by clicking the bottom right button to “Estimate”. There will be the estimated fee in real-time. If the fee is not reasonable, click the “Re-estimate” or wait for a non-congestion period.

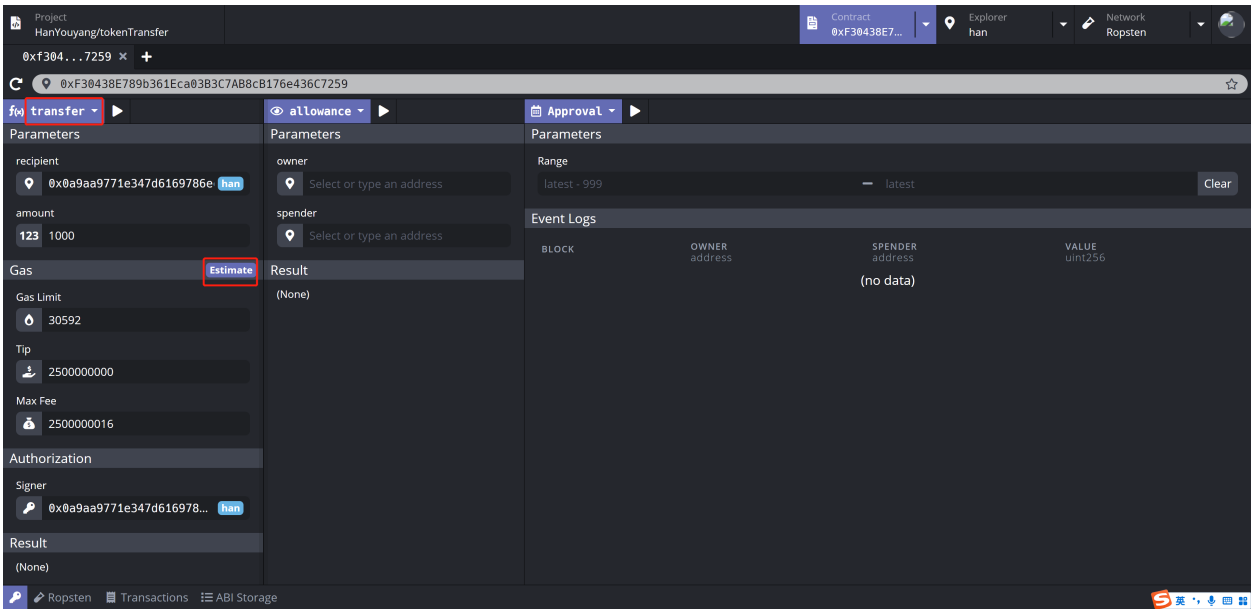


After deploying successfully, a window with detailed transaction information will pop up. Now the contract has been deployed at the address: 0xF30438E789b361Eca03B3C7AB8cB176e436C7259. Click the address to turn to the "Contract" interface.

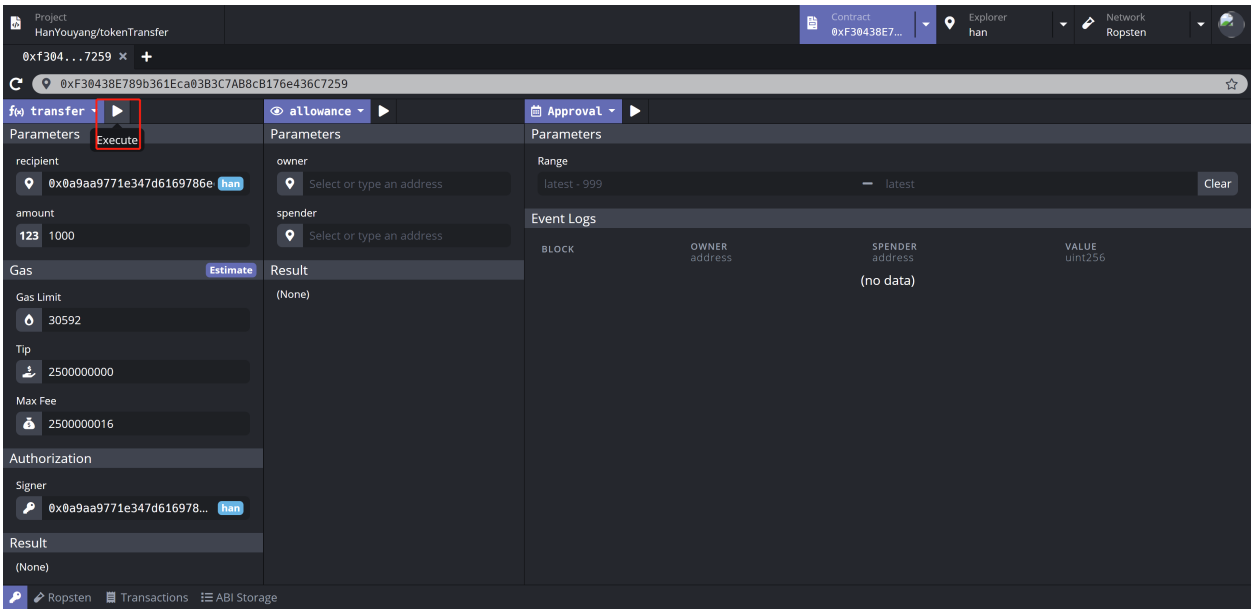


3.7 Check Balance and Transfer

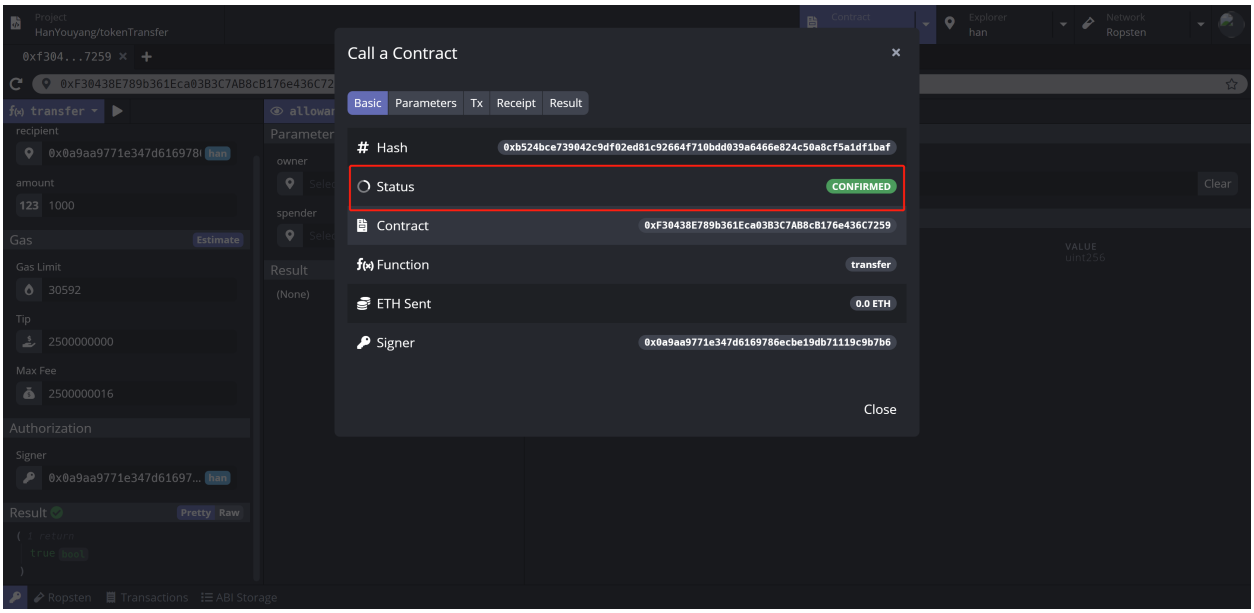
Since 5 test ETH is far enough for paying fees, choose the "transfer" function from the purple inverted triangle icon. Select the "recipient" as the target address and set the token amount to transfer as 1000. Click the purple "Estimate" button to set "Parameters" and "Gas" automatically.



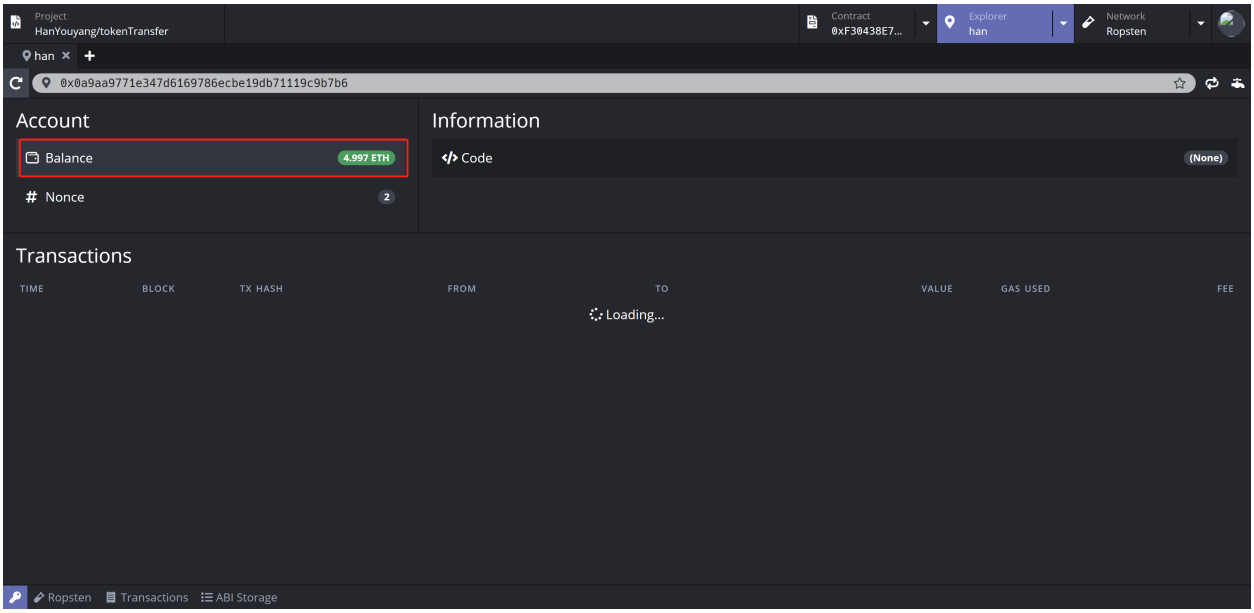
After all the parameters are settled, click the triangle icon beside "transfer" to execute a function.



After the compeletion of "PUSHING" state, the contract is deployed successfully with the "CONFIRMED" state. Select "Explorer" on the upper right panel to see the past transaction detail.



After transferring the 1000 "TEST" token, click the upper right "Explorer" to see the left balance of 4.997 test ETH now. The balance consumed by the "Gas Fee" and "Tip" will not be 5 testETH for deploying and calling a contract.



Here is the most simple quickstart example of the Ethereum Studio. Please feel free to ask us any questions through Github issue link.

CHAPTER

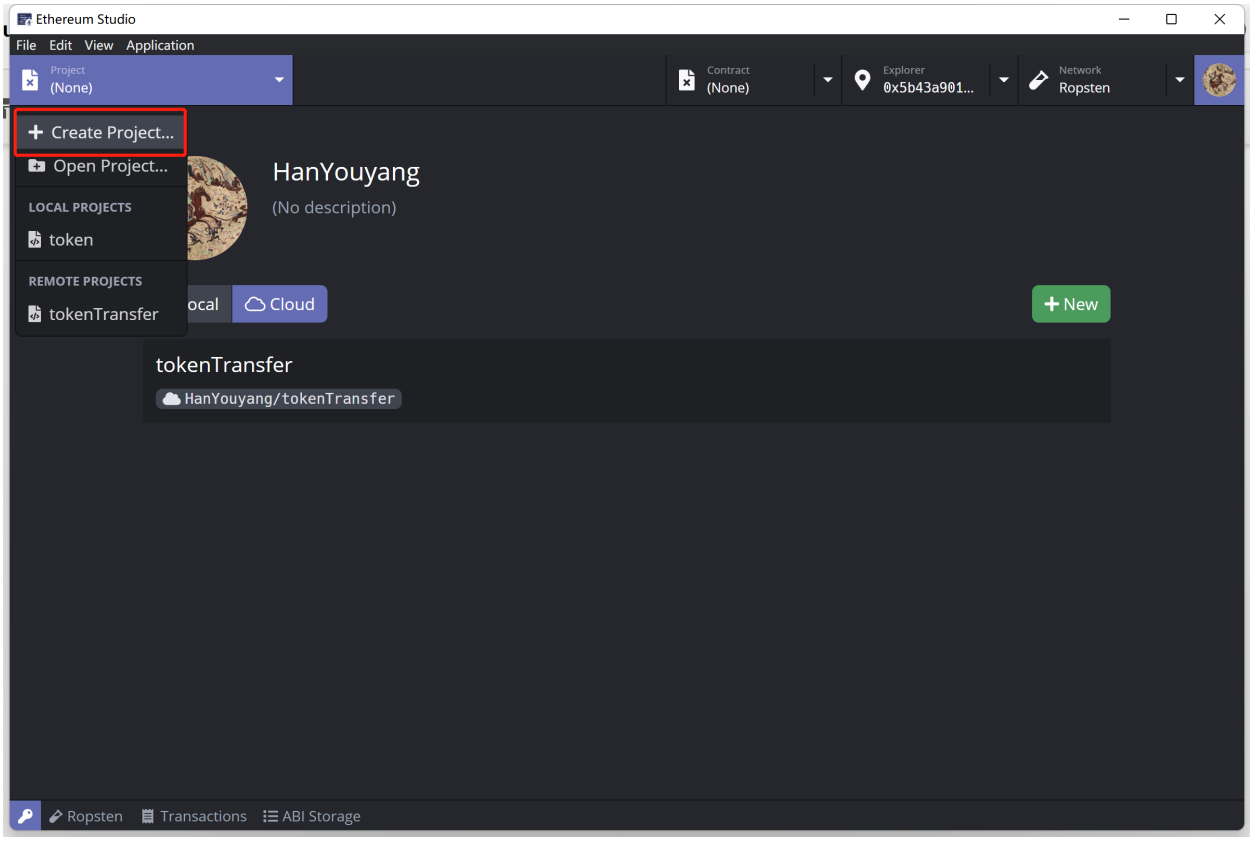
FOUR

PROJECT

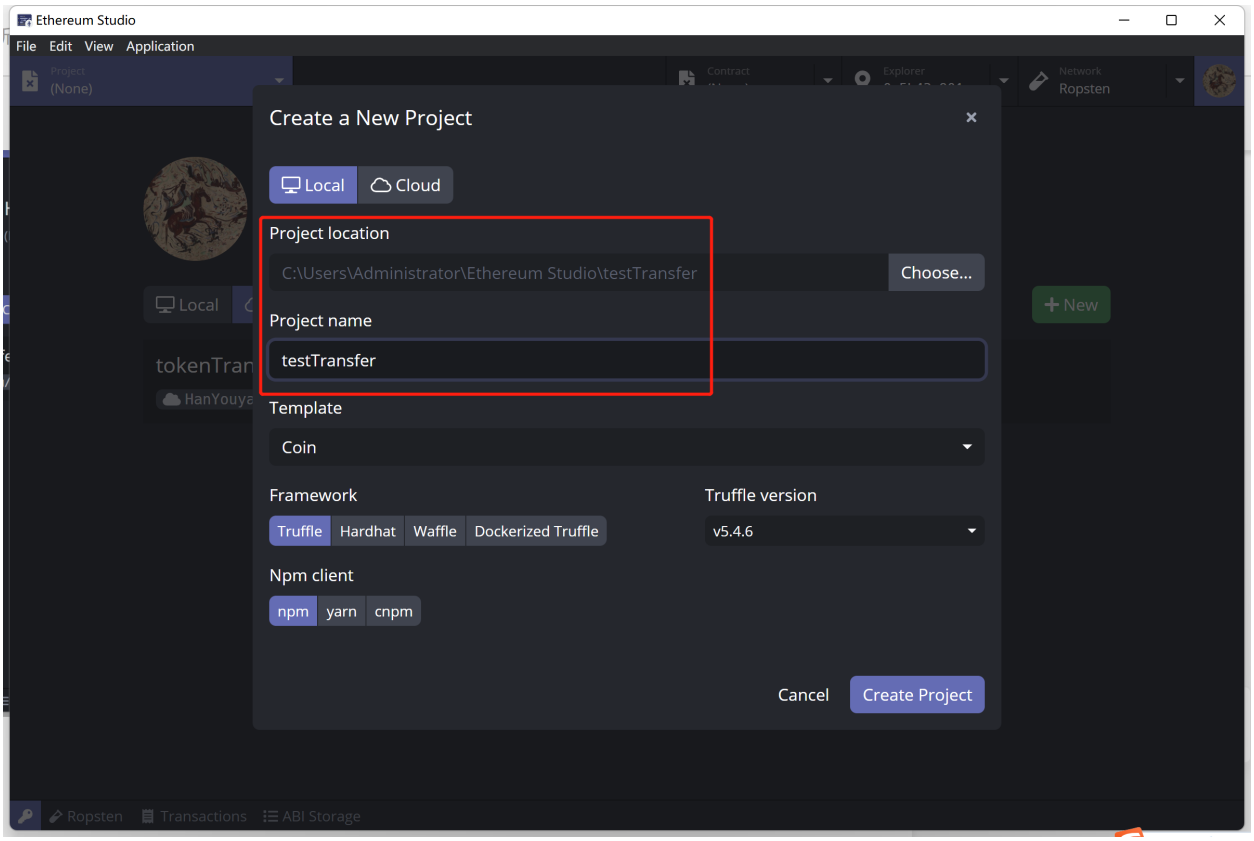
Developers can create a new project, open projects and check all the local and remote projects in the upper left corner "Project" panel.

4.1 Create Project

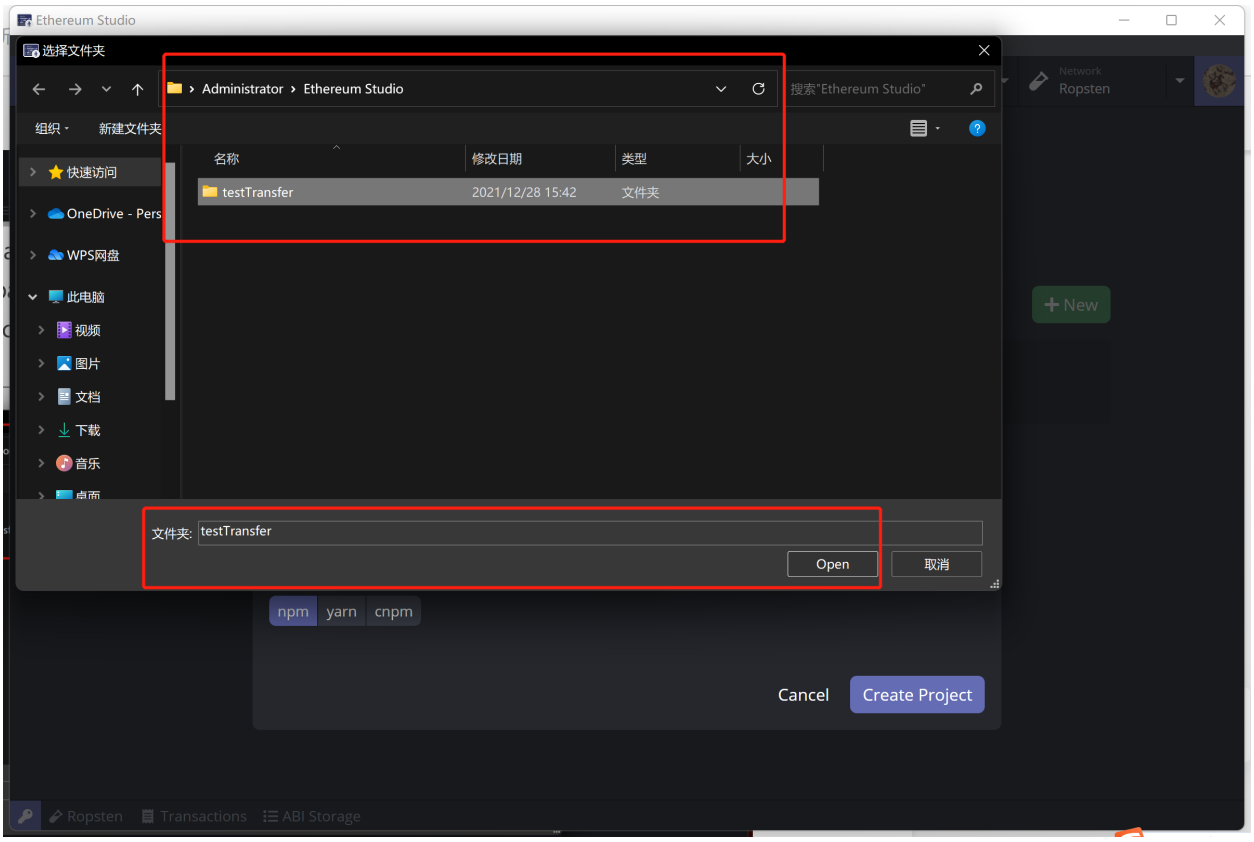
Click "Create Project" and create a new project in either local or cloud under account. The created projects will show in both desktop and web clients in real-time. Creating a project and editing it can be done solely in the Ethereum Studio without other development tools like Visual Studio Code.



In "Create a New Project" panel, developers input the "Project name", while the "Project location" will be automatically generated a document path following "C:\Users\Administrator\Ethereum Studio" in Windows.



Developers can change the path to any other documents. Click the “Choose” button on the left of the default path, and one will see all documents in the whole computer and choose another existing document or create a new document as the new project location. Be careful that the new project location must be an empty file.



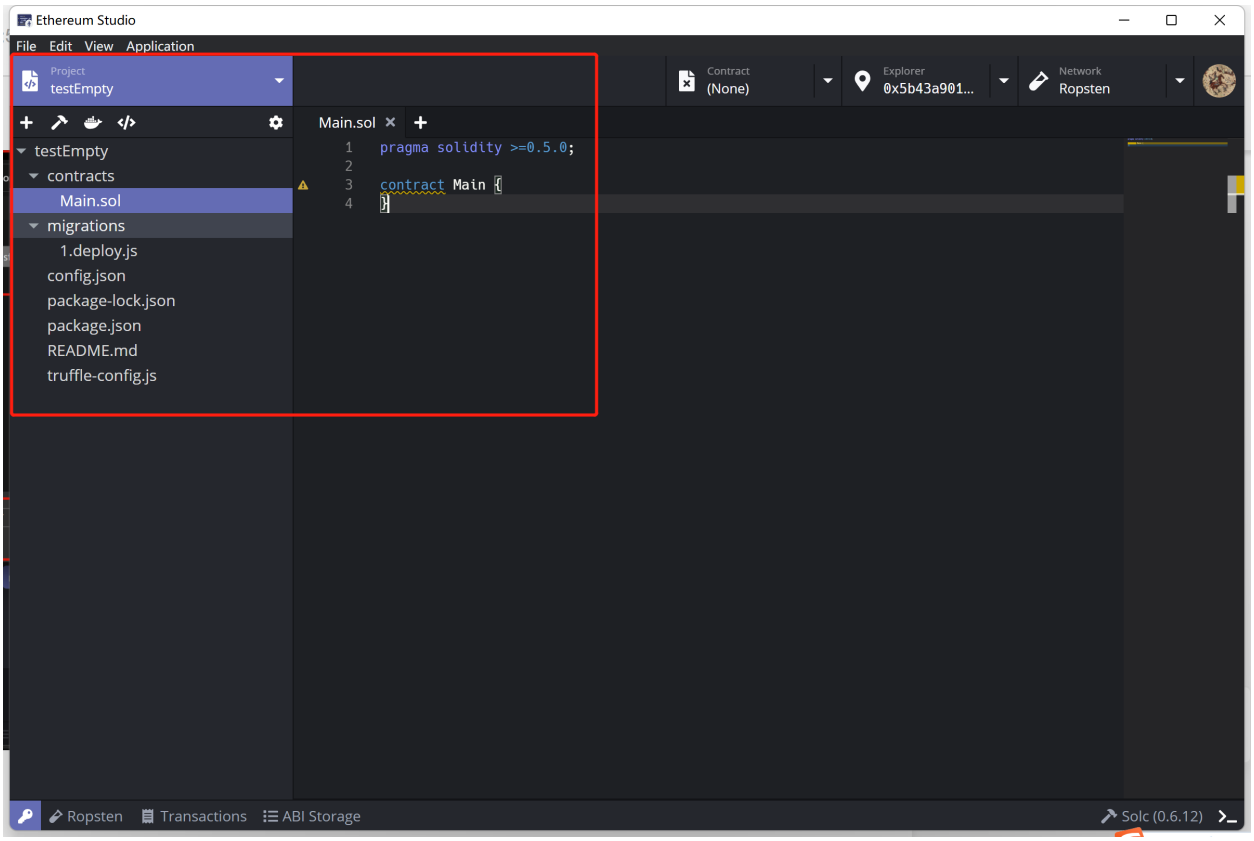
4.1.1 Frameworks

Developers can choose 4 different frameworks: Truffle, Hardhat, Waffle and Dockerized Truffle. The detail information about those frameworks can be checked in "Supported Frameworks" under "Reference" of this doucument.

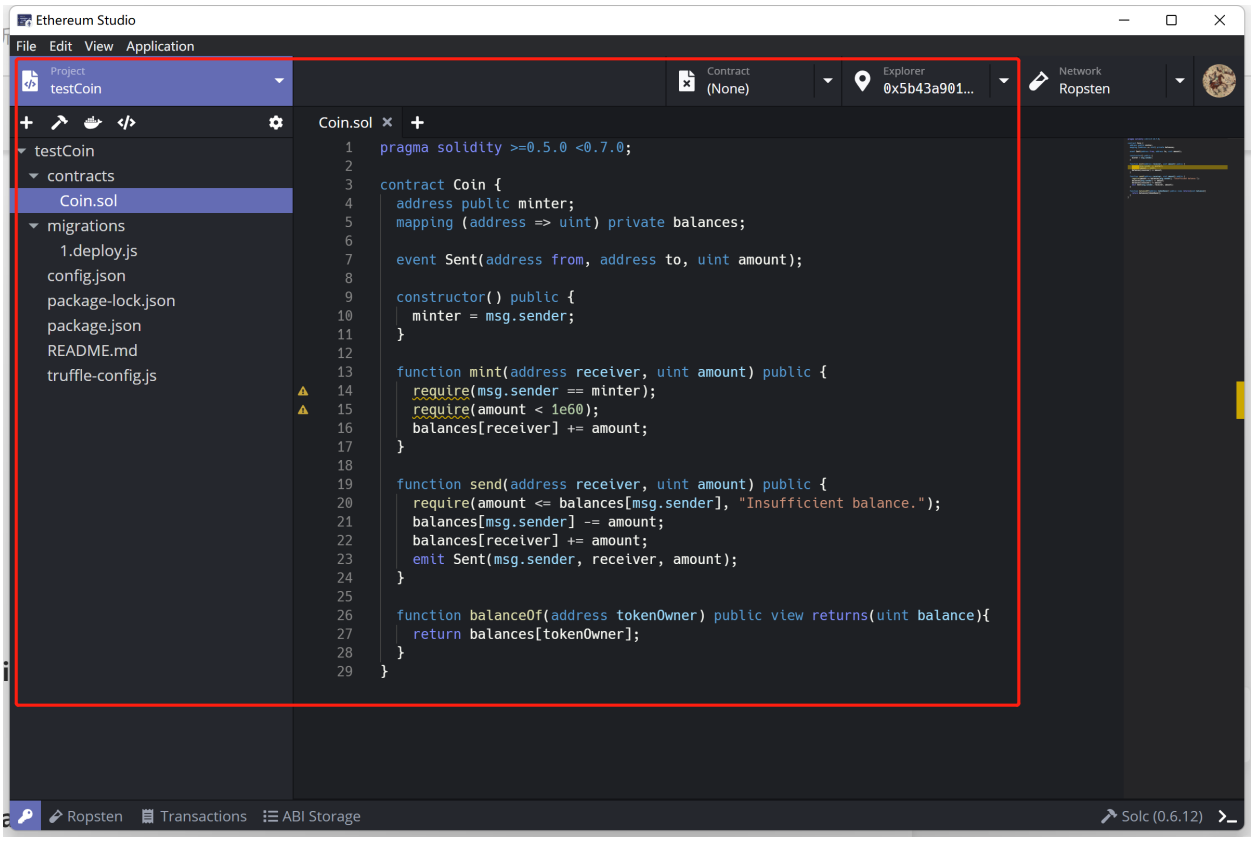
4.1.2 Template

In "Template", there are several different templates including "Empty", "Coin", "ERC20", "Basic" and "Metacoin".

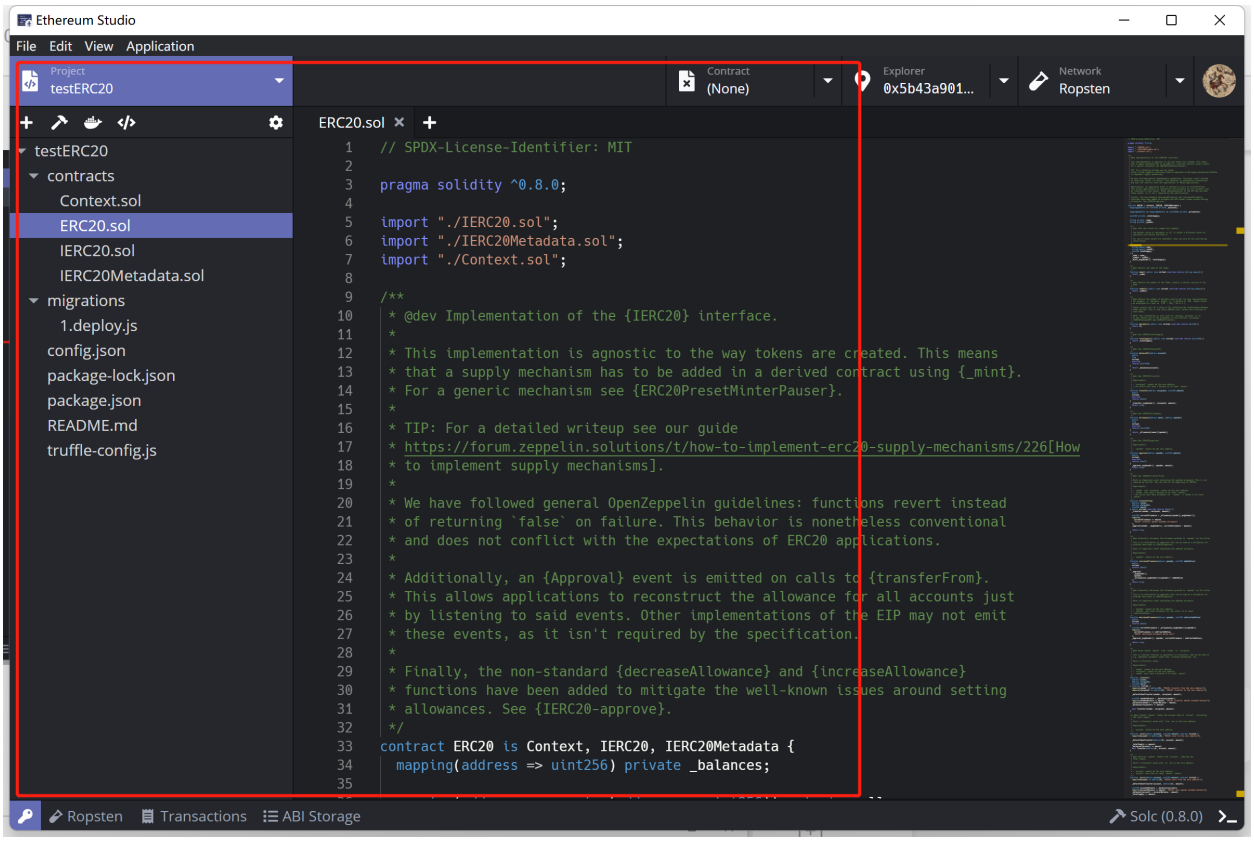
In "Empty Project" template, there is a default empty project with contracts "Main.sol". The only "Main.sol" contract has no content. This template is a minimum viable product to build smart contracts from scratch.



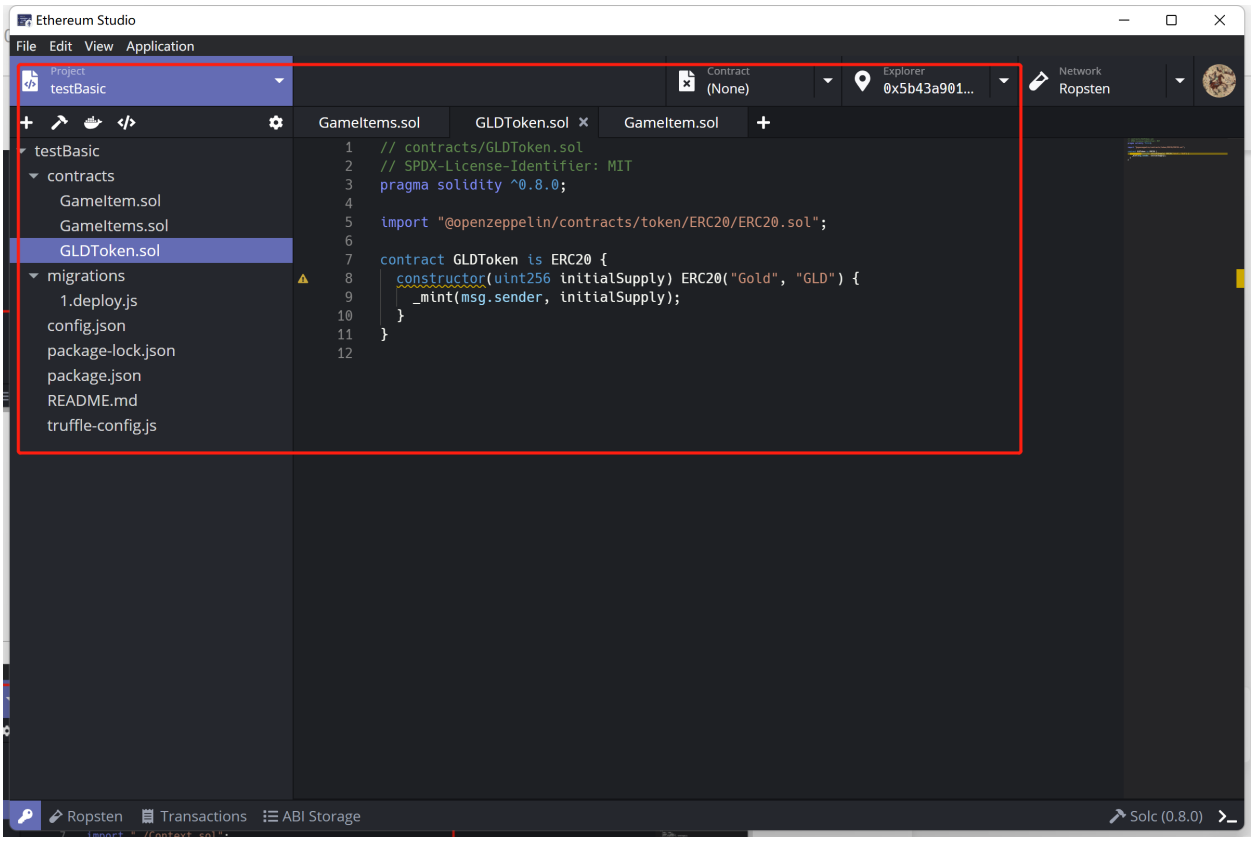
In “Coin” template, there is a default empty project with contracts “Coin.sol”. This template is a primary contract with variables and functions that could help developers build coin-related contracts. Please remember that a definition of coin functions creates this contract and it does not follow any ERC standards.



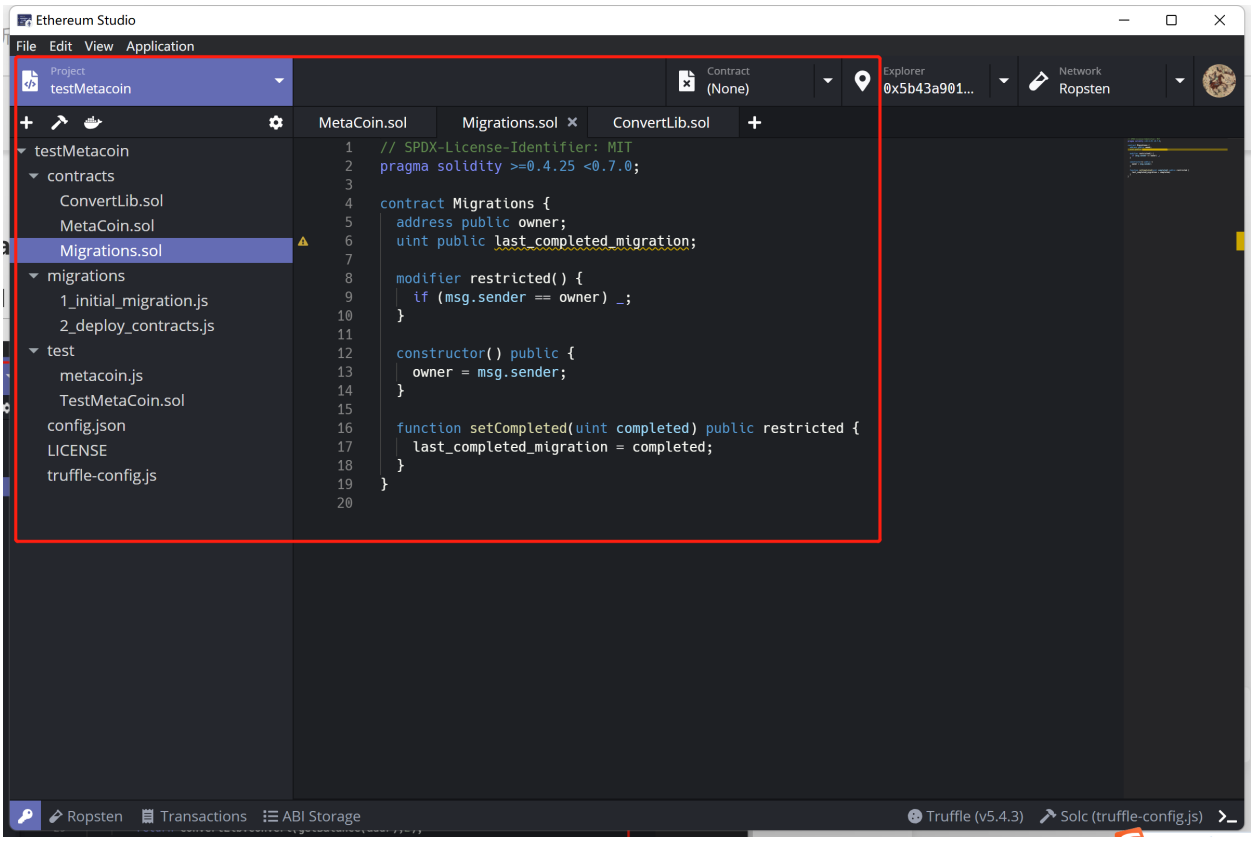
In “ERC20 token” template, there is an ERC20 standard token project including “ERC20.sol”. The “ERC20.sol” contract has all functions the ERC20 standard required. There are also “IERC20” interface. Using the interface would reduce work of setting parameters and provide reliable third-party essential functions for developers to call. This template and generates ERC20 standard related contracts and interfaces locally.



In “Basics - ERC20, ERC721 & ERC1155(v3.1+)” template, there are three ERC standard contracts including “GLDToken.sol”, “GameItem.sol” and “GameItems.sol”. Those three smart contracts inherited ERC20, ERC721 and ERC1155 standards respectively. All three projects import ERC standards through Open Zeppelin and realized only constructor function with basic parameters. Developers can use those smart contracts to understand how the ERC20 tokens and ERC721 and ERC1155 NFTs are minted. The “Basic OpenZeppline Template” will use interfaces online and developers only need to import ERC standard contracts through code.



In “Metacoin” template, there is a default framework “Dockerized Truffle” including “ConvertLib.sol”, “MetaCoin.sol” and “Migrations.sol”. In this project, developers choose “Dockerized Truffle” since ???



4.1.3 npm Clients

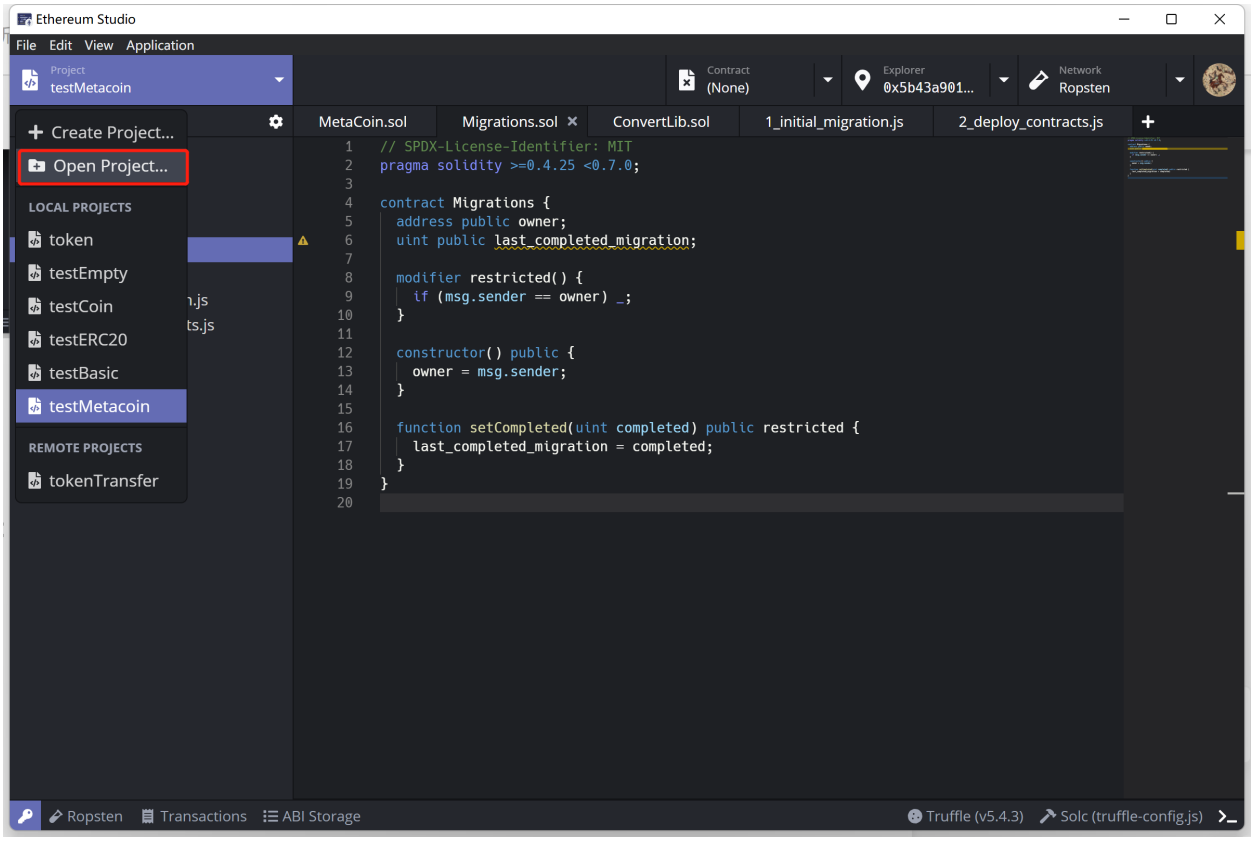
npm(Node Package Manager) stems from when npm first was created as a package manager for Node.js. All npm packages are defined in files called package.json. The content of package.json must be written in JSON. At least two fields must be present in the definition file: name and version.

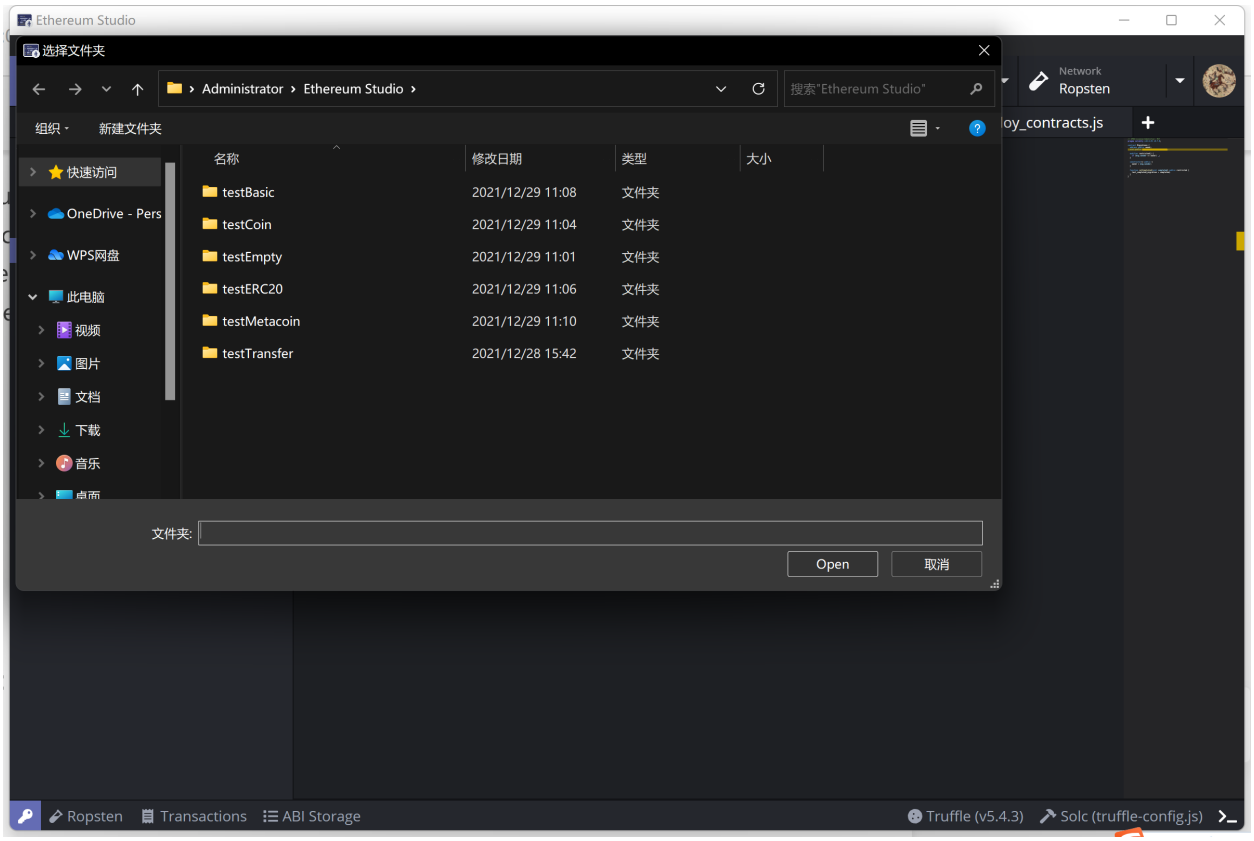
cnpm is faster than npm in China, because Taobao first requests the contents of foreign servers to its own domestic servers, so when we use cnpm, the download depends on downloading from domestic servers, which is much faster. It has a complete image of npmjs.org. At present, the synchronization frequency is once every 10 minutes to ensure that it is synchronized with the official service as much as possible.

yarn offers offline mode. If developers have installed a package before, developers can install it again without any Internet connection. Yarn has a lock file that records the exact version number of the installed module. Each time a file is added, yarn will create (or update) the yarn.lock file to ensure that the module version is the same each time the dependency is installed. Yarn reduce the different versions of dependent packages to a single version to avoid creating multiple copies.

4.2 Open Project

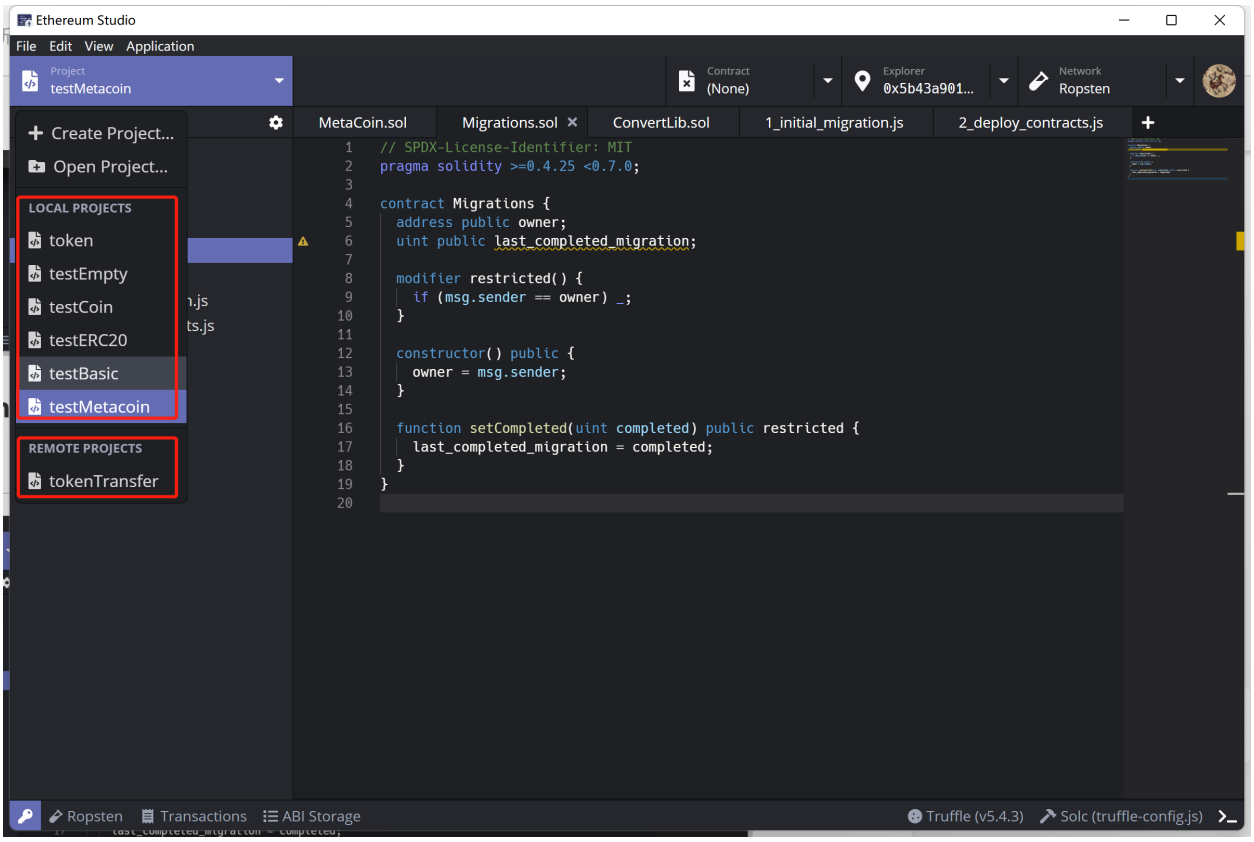
Click “Open Project”, one can check the location of the current project or open an existed project in the system.





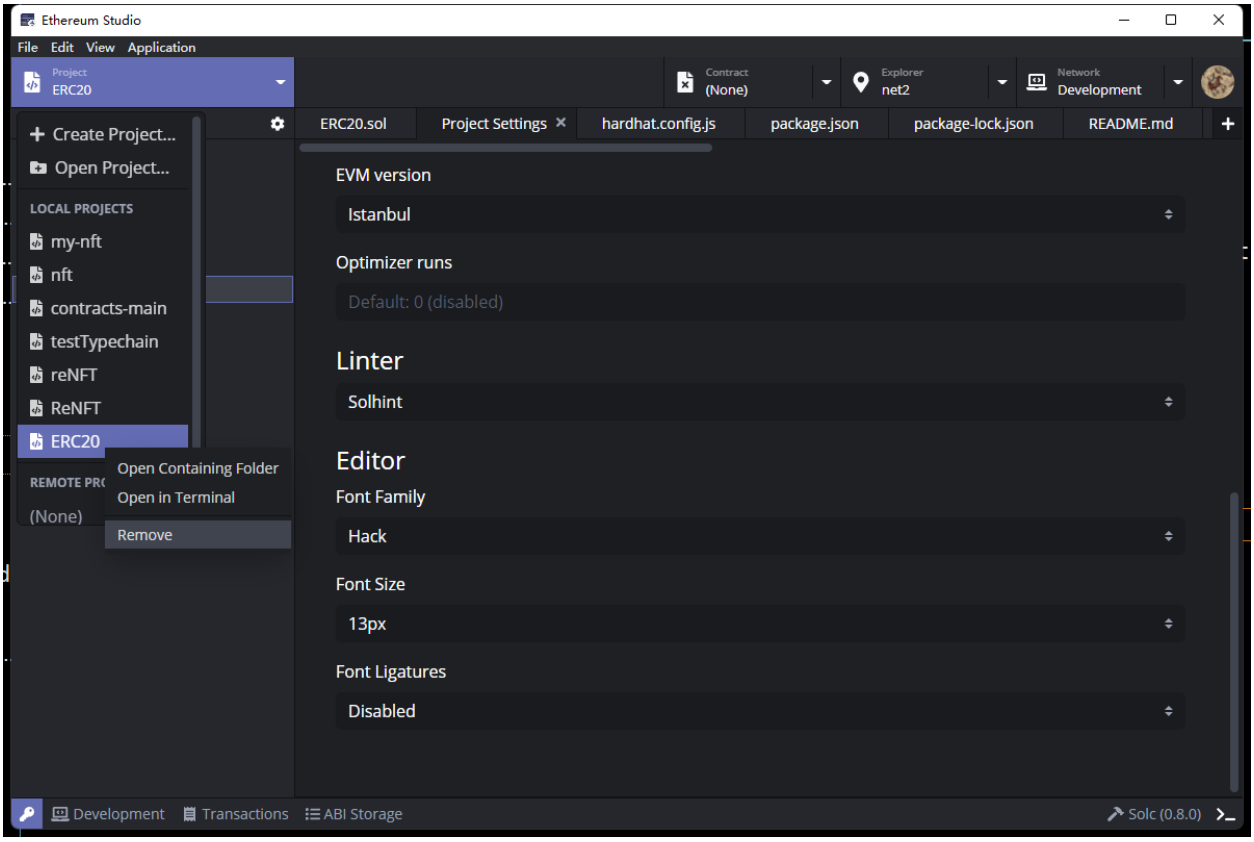
4.3 Local Projects and Cloud Projects

In the “Local Projects” and “Remote Projects” of the “File” panel, the user can see all the projects in local computer and cloud under one’s account. Developers can quickly switch to another project through panel that is helpful for muti-project developers.



4.4 Delete Project

Right click the name of any project in the panel, developers can chose “Remove” and the project is deleted immediately. Be careful since the deleted project can not be restored from “Trash Can” on the desktop.


Ethereum Studio
File Edit View Application
Project
ERC20
Contract
(None)
Explorer
net2
Network
Development
Create Project...
Open Project...
LOCAL PROJECTS
my-nft
nft
contracts-main
testTypechain
reNFT
ReNFT
ERC20
REMOTE PR
(None)
Open Containing Folder
Open in Terminal
Remove
ERC20.sol
Project Settings
hardhat.config.js
package.json
package-lock.json
README.md
EVM version
Istanbul
Optimizer runs
Default: 0 (disabled)
Linter
Solhint
Editor
Font Family
Hack
Font Size
13px
Font Ligatures
Disabled
Development
Transactions
ABI Storage
Solc (0.8.0)

CHAPTER

FIVE

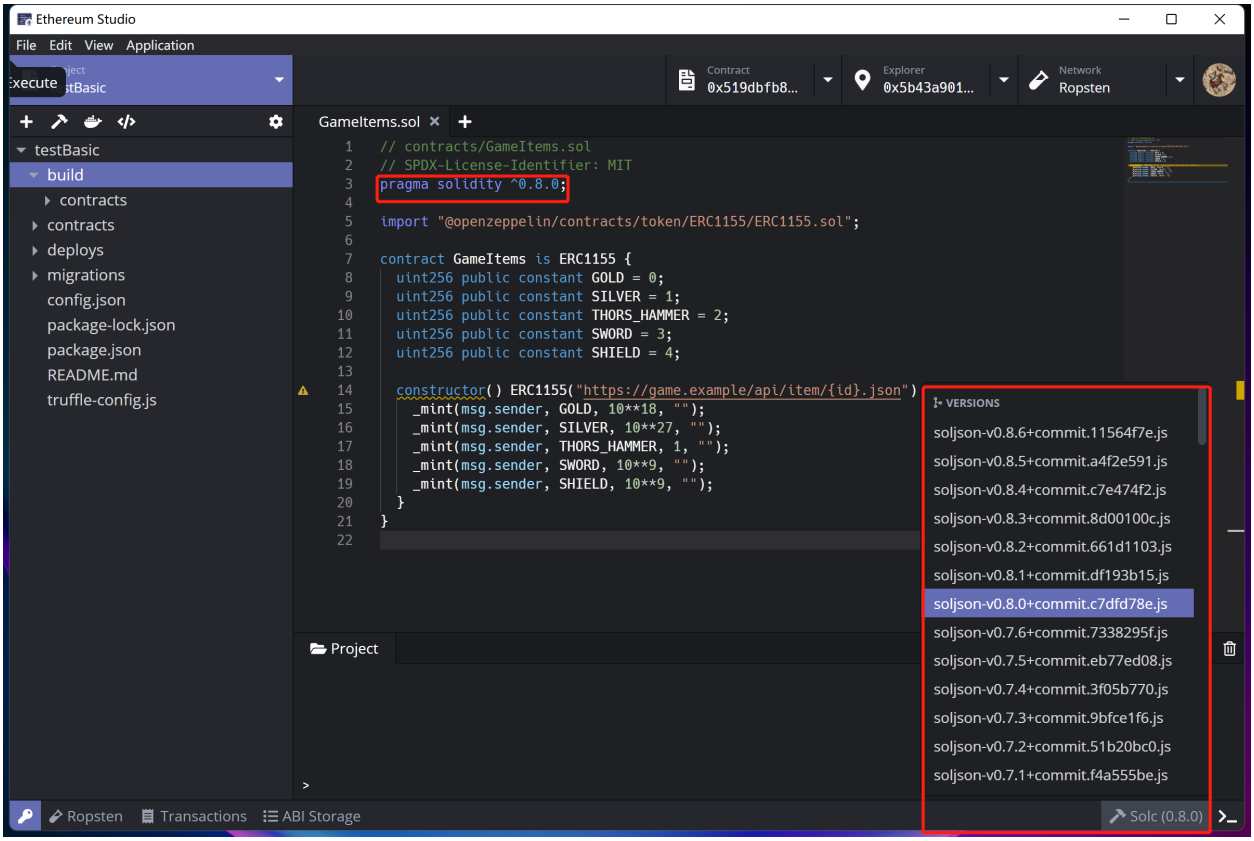
EDITOR

5.1 Build

After test and develop, developers needs to build smart contracts.

5.1.1 Build Preparation

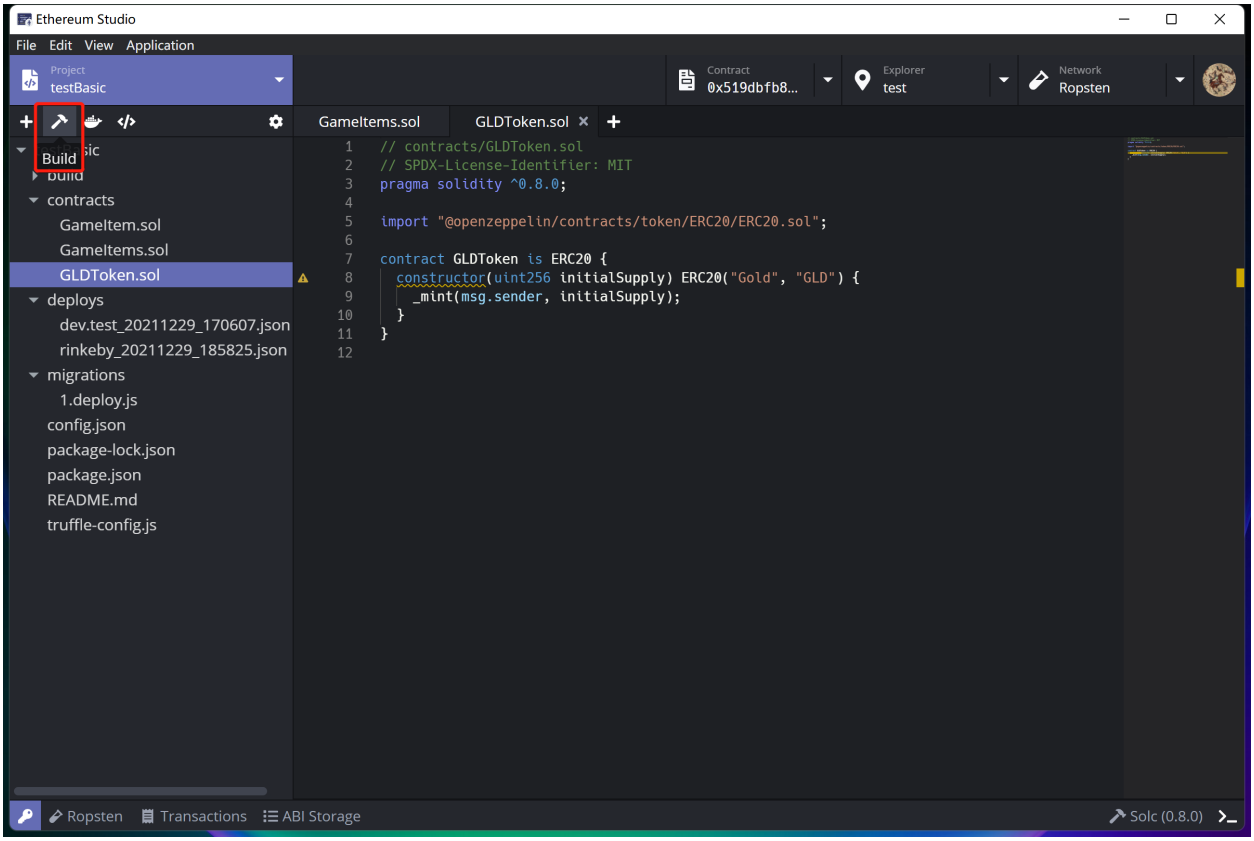
When an existed project is opened, the Ethereum Studio can automatically detect the solidity version written at the head of contracts. Developers can select the specific version of the solidity compiler by clicking the “hammer” icon at the bottom right corner with “Solc(0.x.y)”.

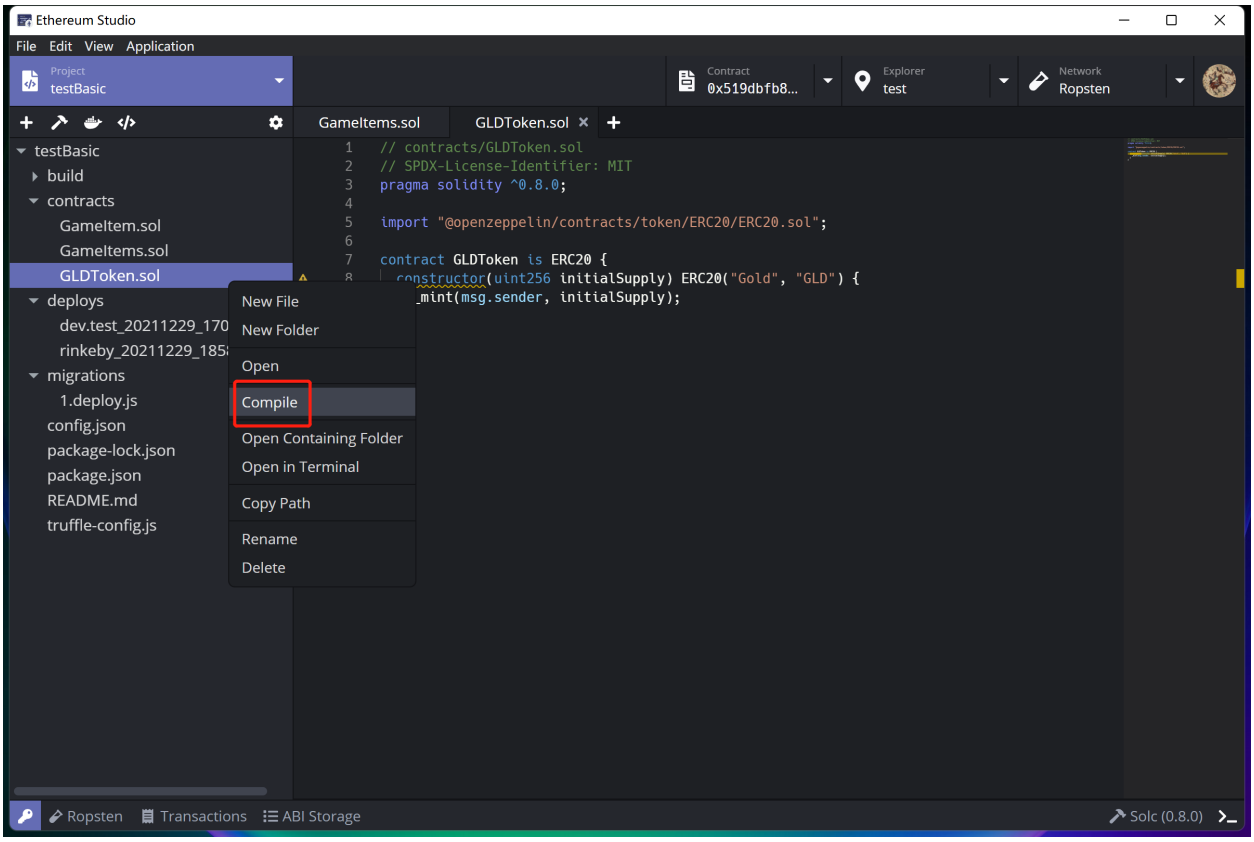


5.1.2 Build by Panel

In the desktop client, developers can click the "hammer" icon below the "Project" panel or right-click the target file and select "Compile" to build the contract. Now the Ethereum Studio only supports building all contracts together, and the single file building will be released later.

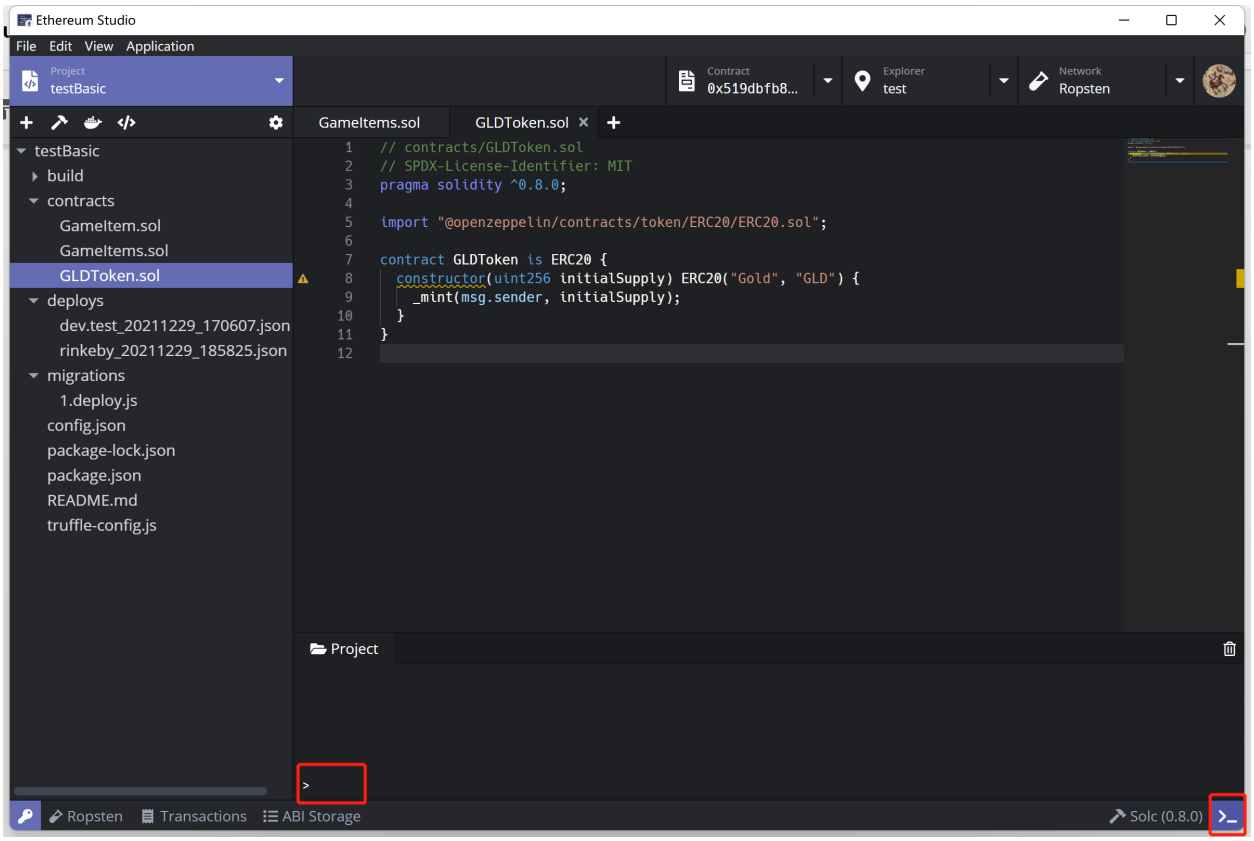
Developers can find differences between web and desktop clients on building a project. The detail explanation can be checked in **Chapter Install Desktop Client, Contrast with Web Client**.





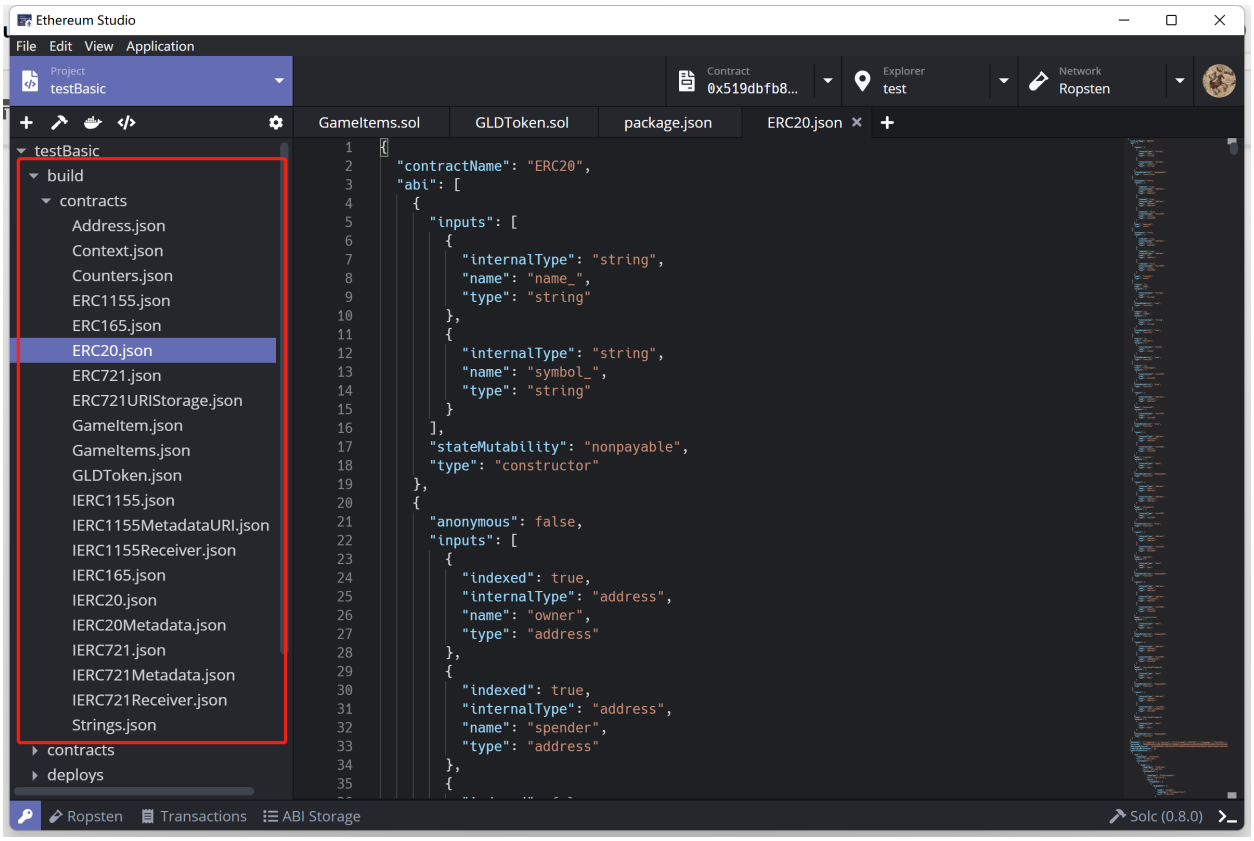
5.1.3 Build by Command Line

Developers can build contracts manually by opening the command line, clicking the "Terminal" icon and inputting the command through "Project" panel. Please note that the command must be corresponding to the selected framework during creating process. Developers can check the framework type in "package.json" with the commands following "scripts".



5.1.4 Check Building Details

After building successfully, the framework will build a new document named “build” containing a “contracts” document. In “contracts” document, there are all the JSON files generated in the building process. Each JSON file has the corresponding application binary interface(ABI). Developers will deploy ABI files on the network later.

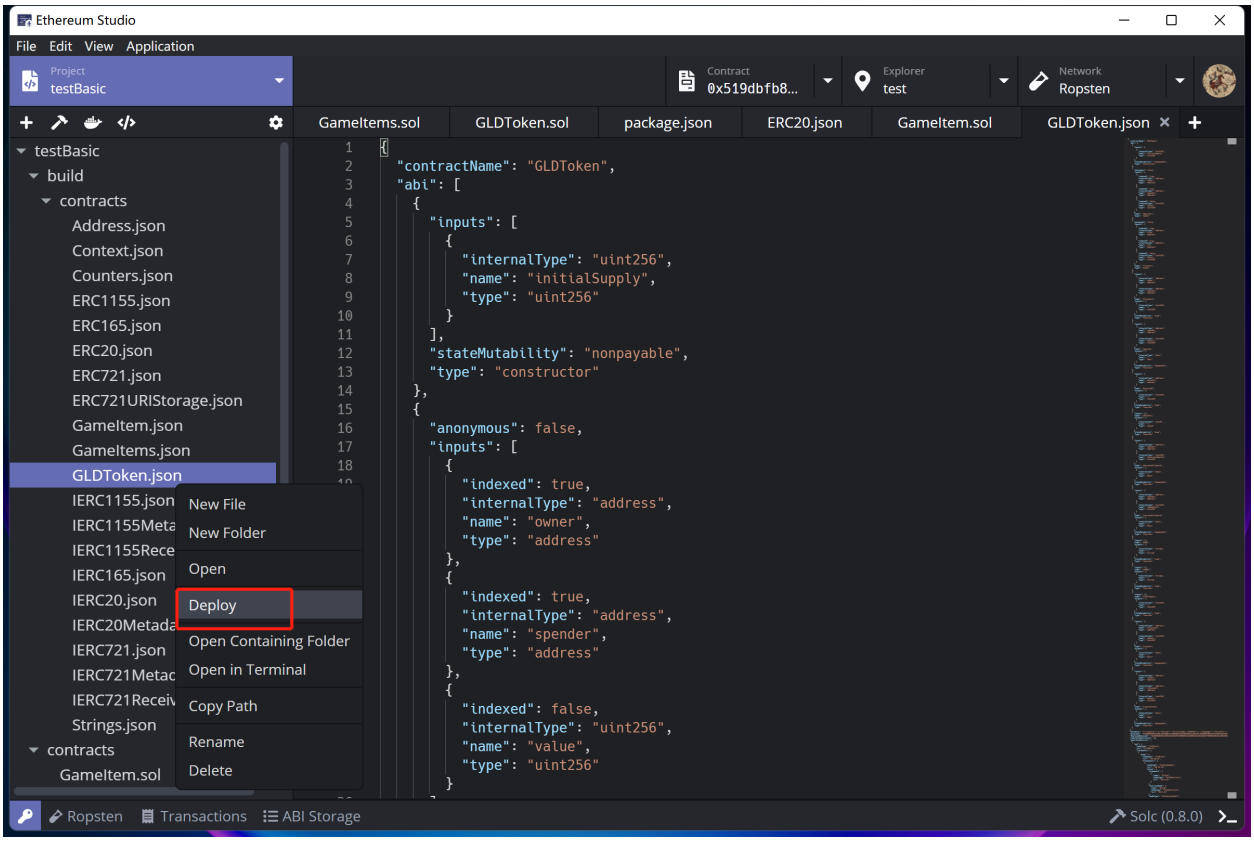


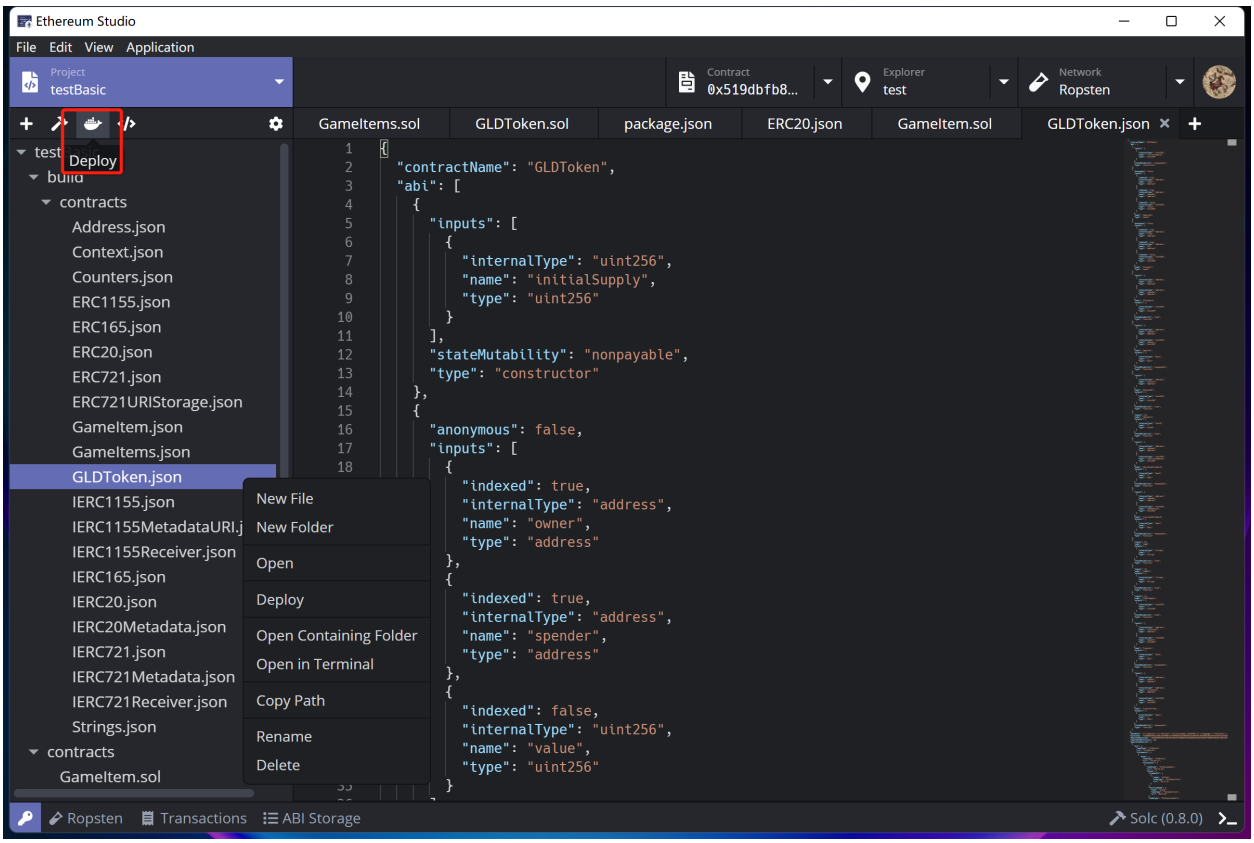
5.2 Deploy

After building process, developers can deploy target contracts with generated ABI files.

5.2.1 Deploy by Panel

Developers can right-click the file name of an contract and select “Deploy”, and there will popup a “Deploy Contract” window. By other means, developers can click the “Docker” icon below the “Project” panel.


Ethereum Studio
File Edit View Application
Project testBasic
Contract 0x519dbfb8...
Explorer test
Network Ropsten
GameItems.sol
GLDToken.sol
package.json
ERC20.json
GameItem.sol
GLDToken.json
testBasic
build
contracts
Address.json
Context.json
Counters.json
ERC1155.json
ERC165.json
ERC20.json
ERC721.json
ERC721URIStorage.json
GameItem.json
GameItems.json
GLDToken.json
IERC1155.json
IERC1155MetadataURI.json
IERC1155Receiver.json
IERC165.json
IERC20.json
IERC20Metadata.json
IERC721.json
IERC721Metadata.json
IERC721Receiver.json
Strings.json
contracts
GameItem.sol
New File
New Folder
Open
Deploy
Open Containing Folder
Open in Terminal
Copy Path
Rename
Delete
Ropsten
Transactions
ABI Storage
Solc (0.8.0)

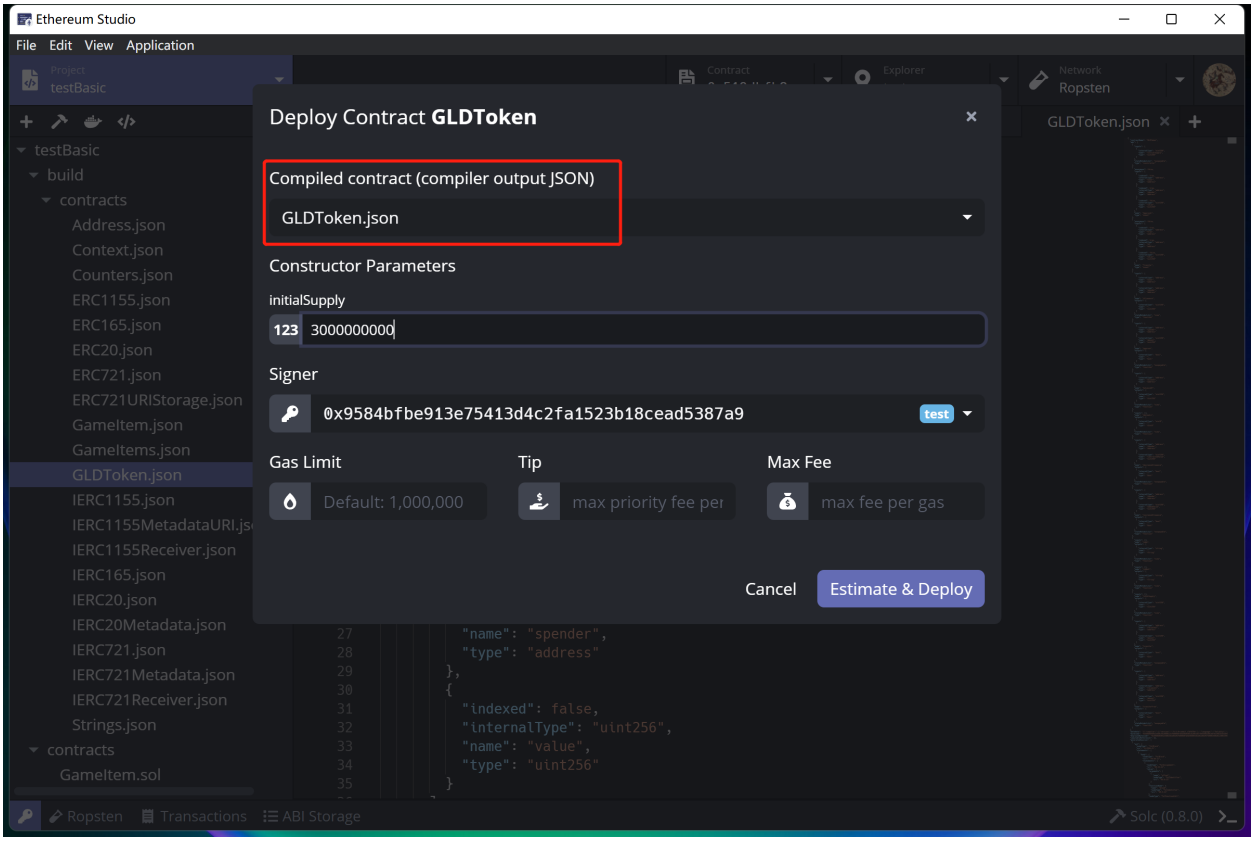

Ethereum Studio
File Edit View Application
Project testBasic
Contract 0x519dbfb8...
Explorer test
Network Ropsten
Deploy
GameItems.sol
GLDToken.sol
package.json
ERC20.json
GameItem.sol
GLDToken.json
contracts
Address.json
Context.json
Counters.json
ERC1155.json
ERC165.json
ERC20.json
ERC721.json
ERC721URIStorage.json
GameItem.json
GameItems.json
GLDToken.json
IERC1155.json
IERC1155MetadataURI.json
IERC1155Receiver.json
IERC165.json
IERC20.json
IERC20Metadata.json
IERC721.json
IERC721Metadata.json
IERC721Receiver.json
Strings.json
contracts
GameItem.sol
New File
New Folder
Open
Deploy
Open Containing Folder
Open in Terminal
Copy Path
Rename
Delete
Ropsten
Transactions
ABI Storage
Solc (0.8.0)

5.2.2 Deploy by Command Line

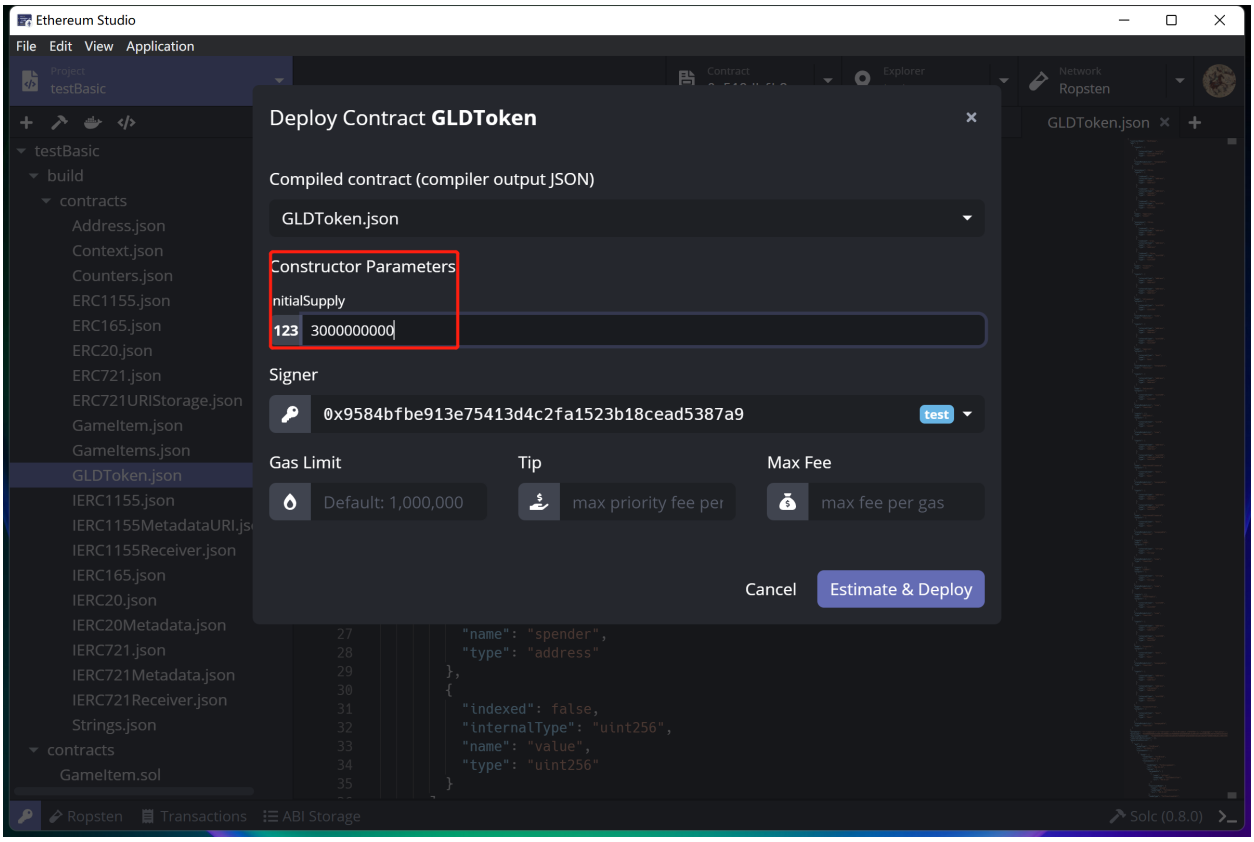
Developers can open the command line to deploy contracts manually by clicking the "Terminal" icon and input command through the "Project" panel. Please note that the commands must be corresponding to the selected framework during project creating process. Developers can check the framework in "package.json" with commands following "scripts".

5.2.3 Deploy Preparation

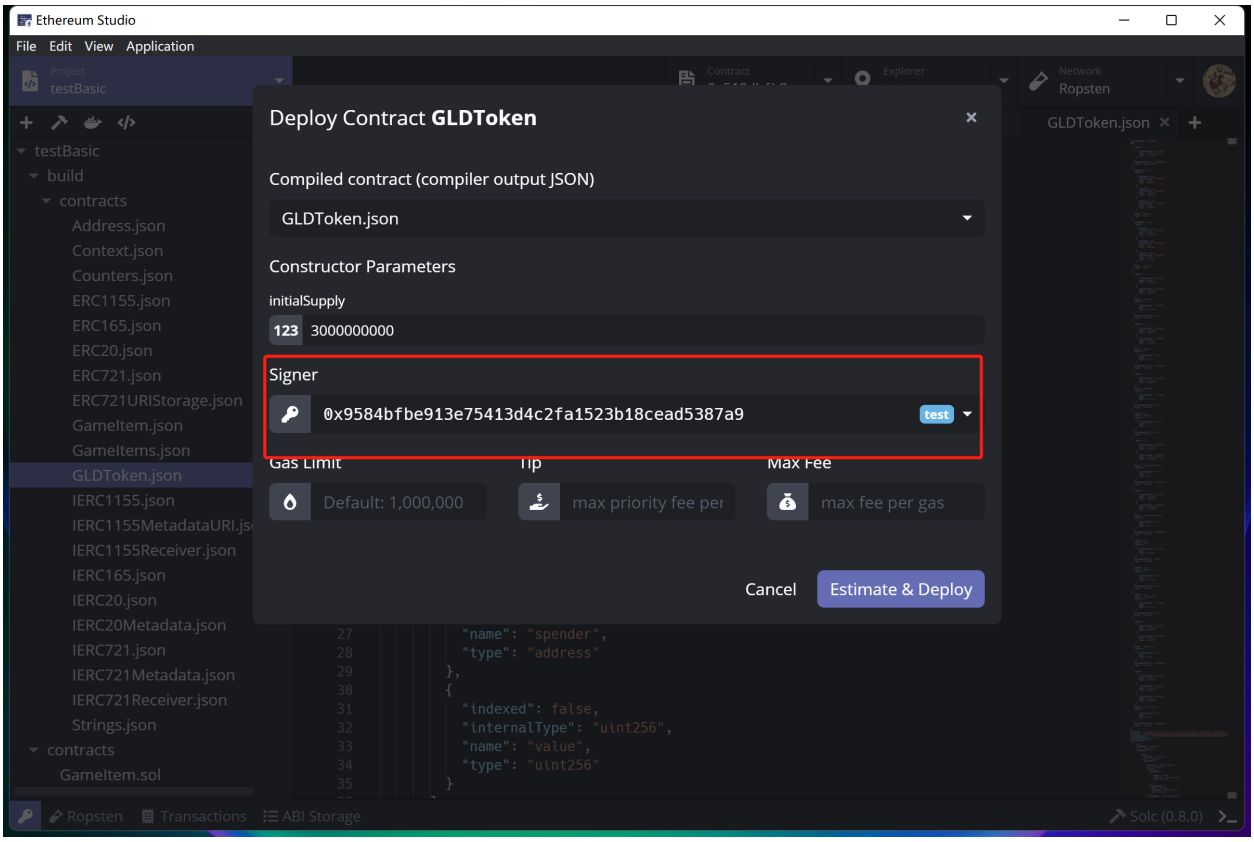
There will be a "Deploy Contract" window in the preparation process. Developers can choose a JSON file to deploy on the network in this window.



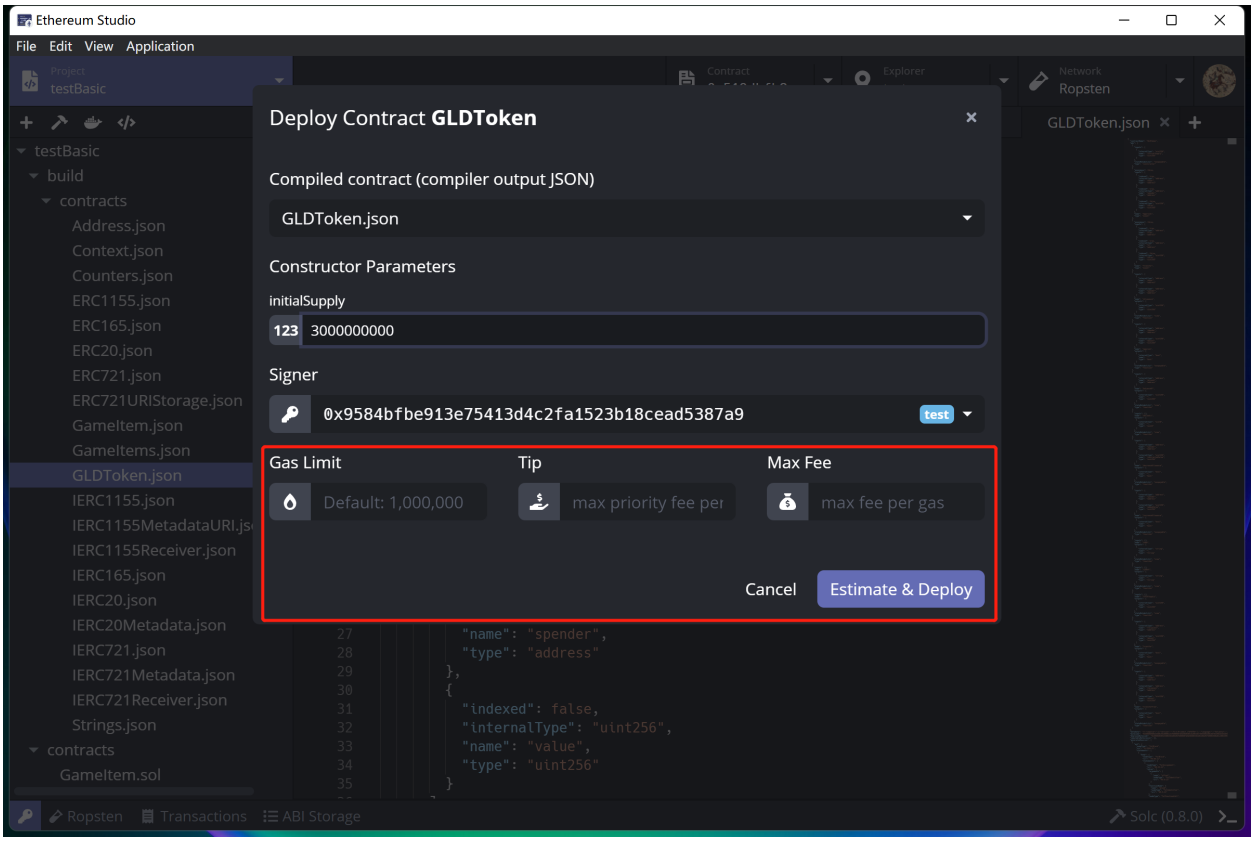
"Constructor Parameters" are corresponding to the constructor function parameters for users to input.



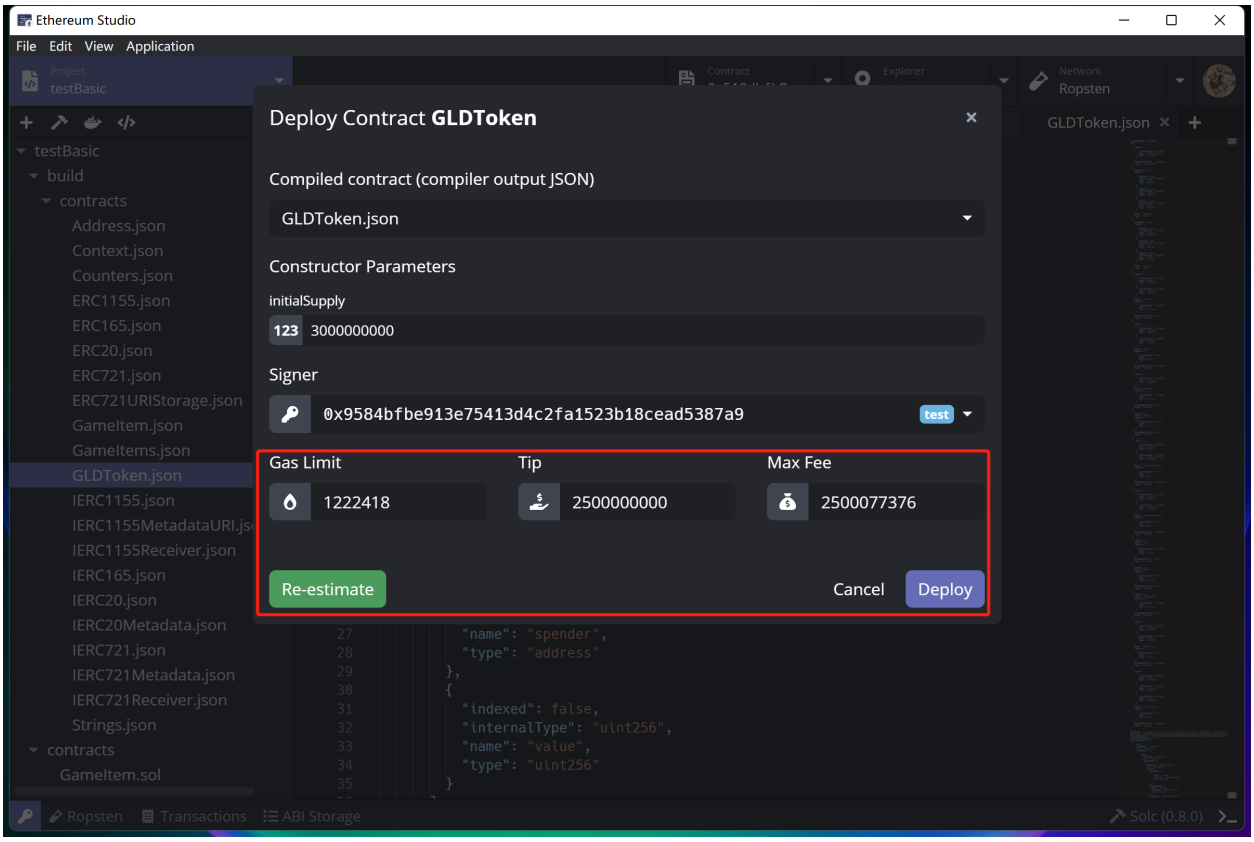
Developers can choose “Signer”, the final payer of gas fee and tip for deploying this contract. Developers needs to ensure the signer has enough ETH on the target network. Otherwise, the later “Estimate & Deploy” process will not be successful.



The "Estimate & Deploy" button locates at the bottom right corner. There will be a real-time estimation of "Gas Limit", "Tip", and "Max Fee" located below. It will be significantly different for the estimation time and price. Developers should carefully check the network gas fee before deploying.

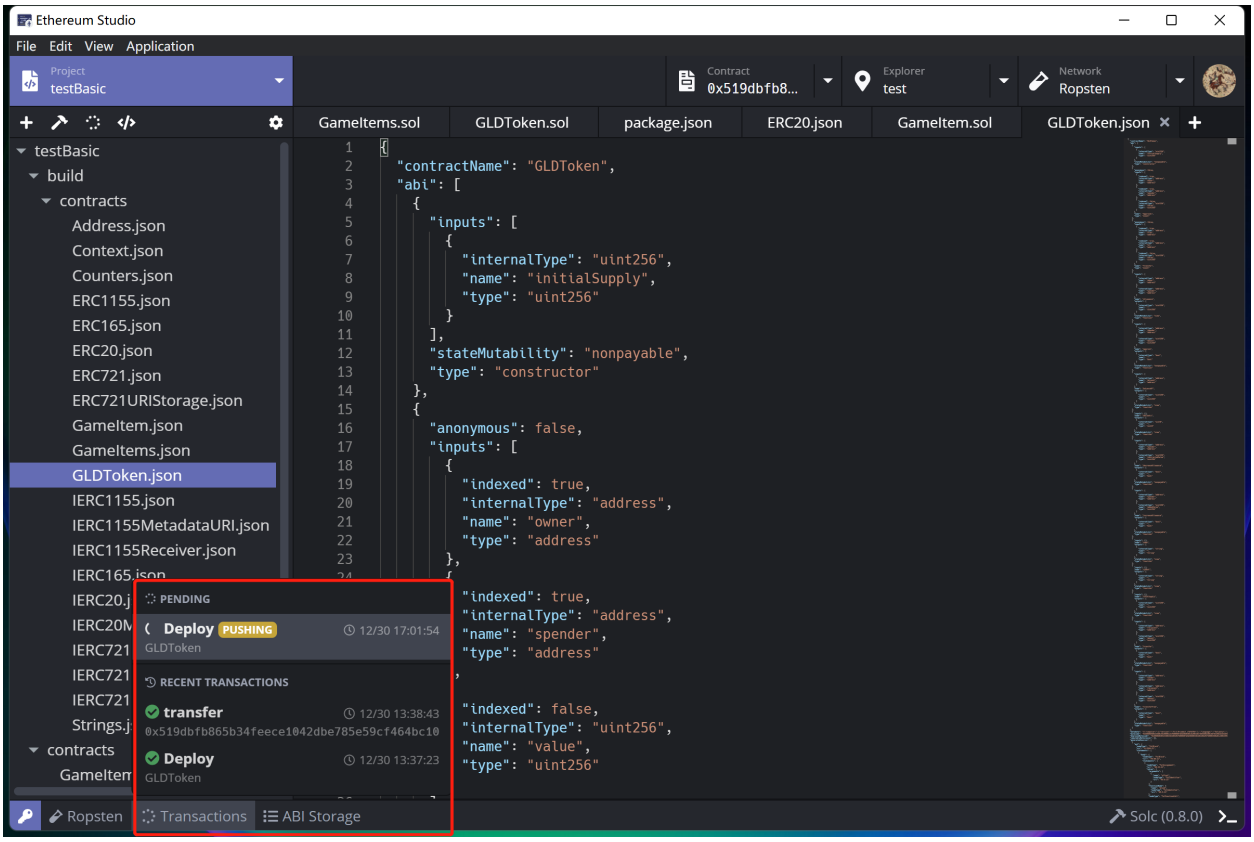


After estimating process, there are exact price numbers in each box. If developers feel the gas fee price is too high or the network is too busy, click the green “Re-estimate” button at the bottom left corner to estimate cost again. If developers supposes the price is fair, click the purple “Deploy” button on the bottom right corner to deploy the contract.

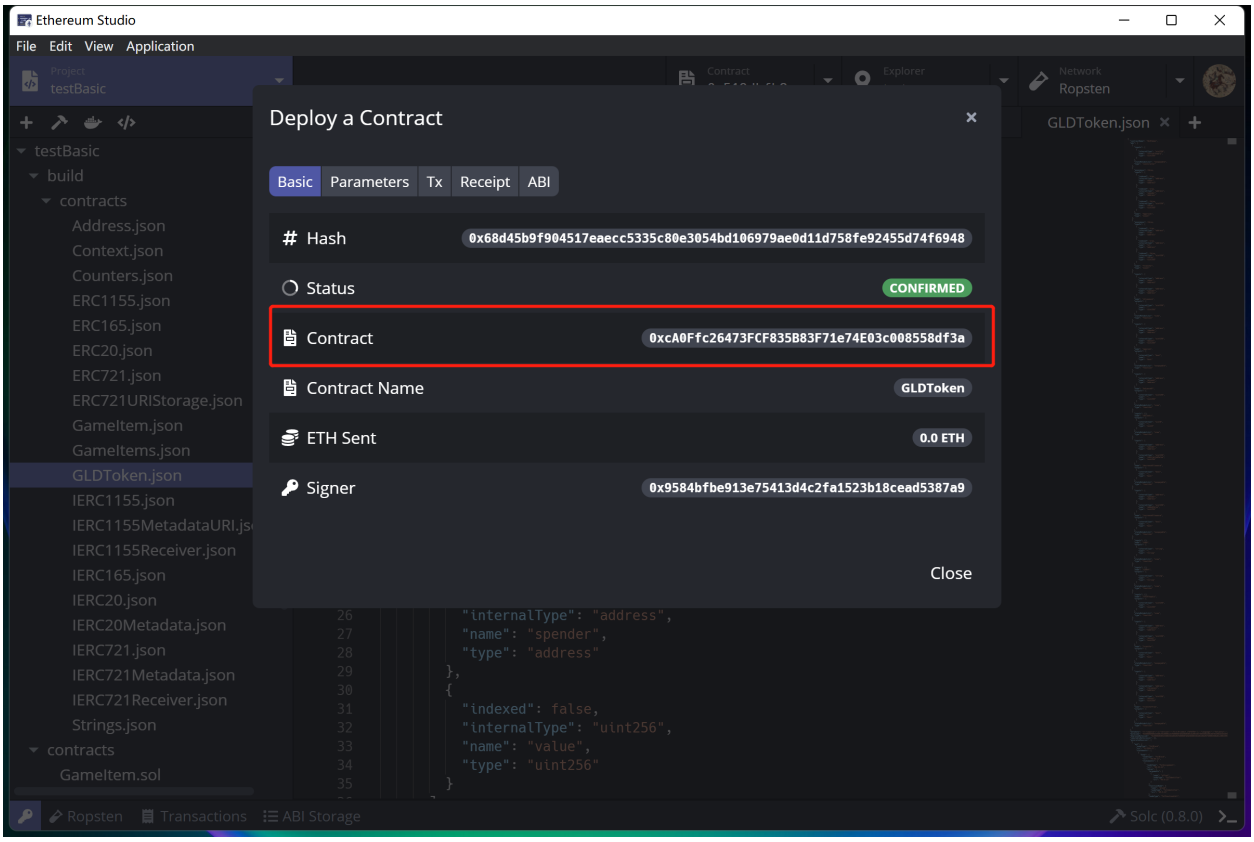


5.2.4 Check Deploy Transactions

After clicking "Deploy", developers can check deploy schedule by clicking the bottom "Transactions" button to review any transaction. Developers can check detailed information with the popup "Deploy a Contract" window.

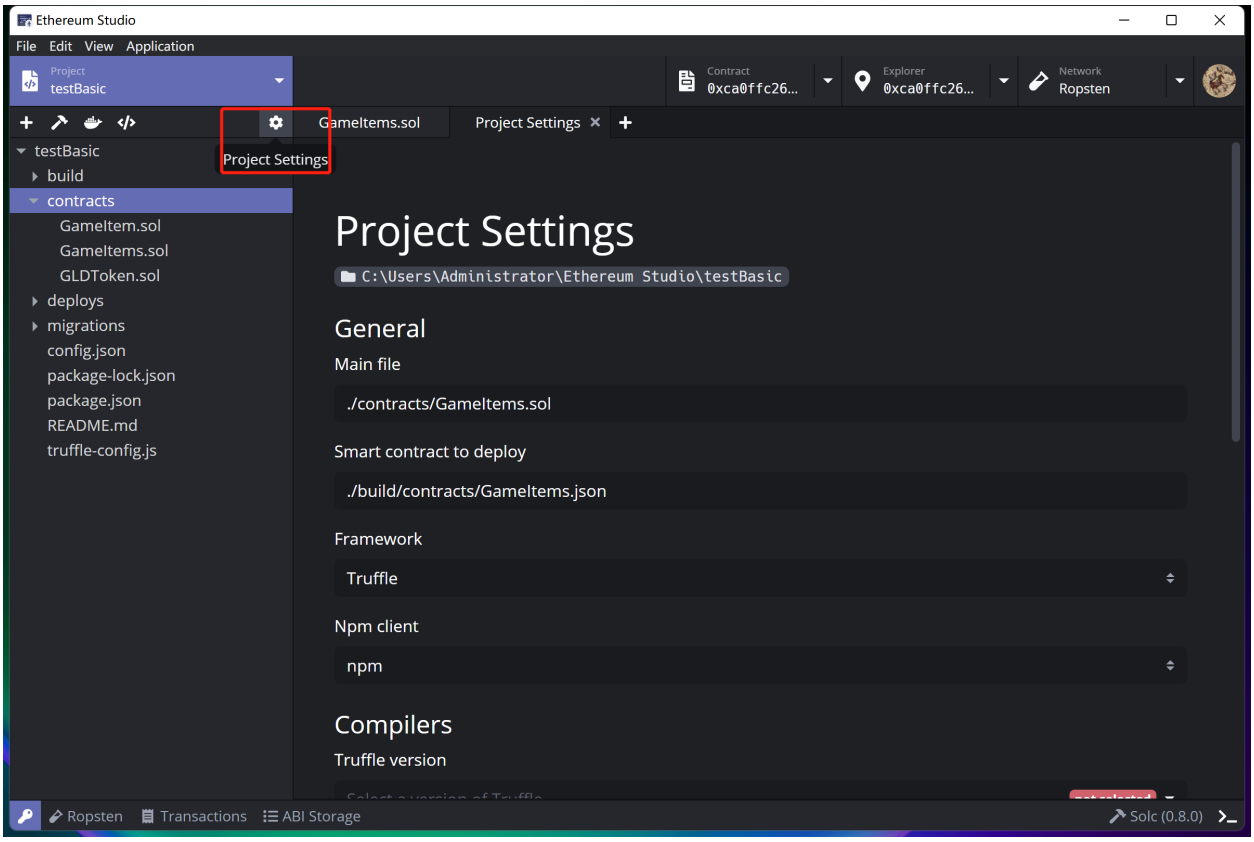


Five boxes show detailed information in the “Deploy a Contract” window. In the “Basic” panel, there are several most crucial pieces of information of deployment, including address. Developers can click on the “Contract” address, and the “Contract” panel will show the contract functions for developers to call.



5.3 Project Settings

At the right end of the toolbar, there is a "gear" icon named "Project Settings". Click the icon, and there will be the "Project Settings" panel in the editor. This panel is a graphic show of the "config.json" file. Developers can easily change settings in the project.

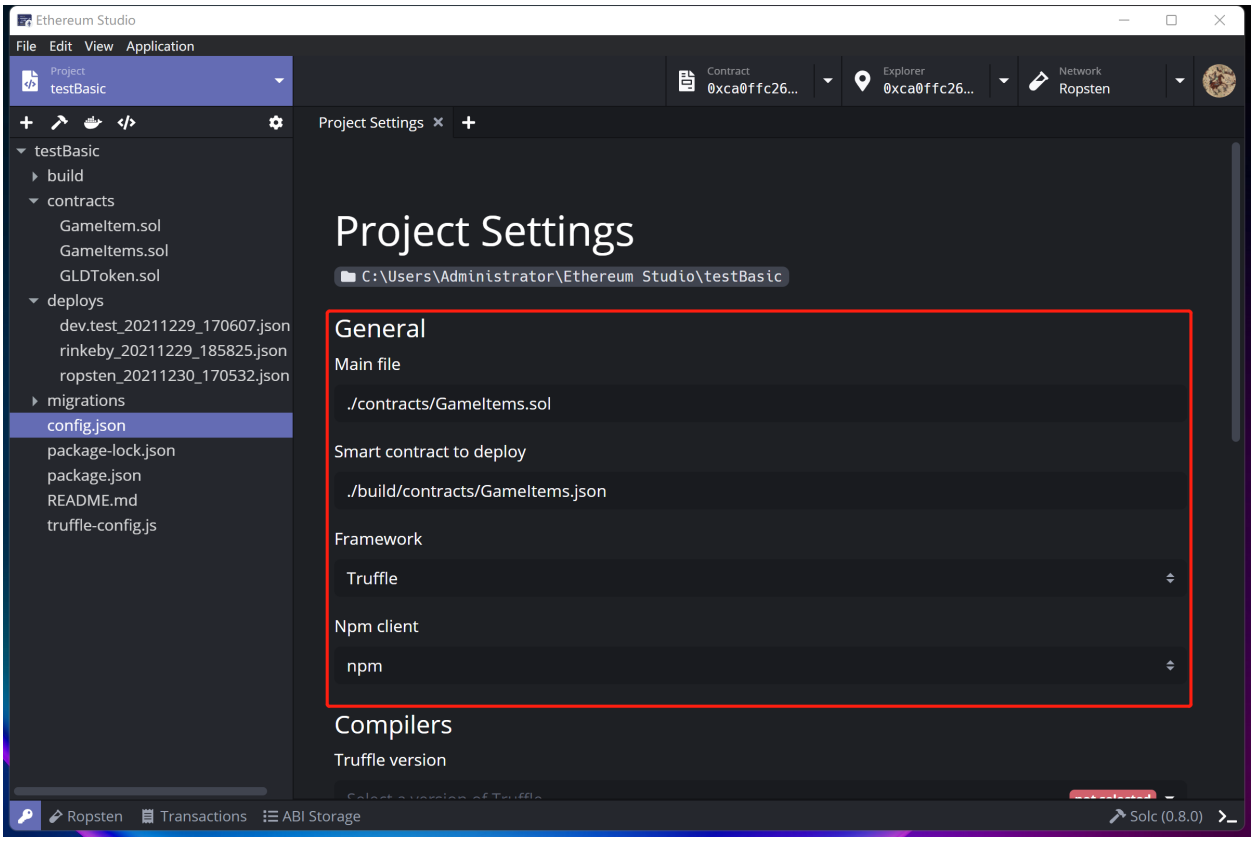


5.3.1 General

In the “General” part, the “Main file” is the default selection of deploying file.

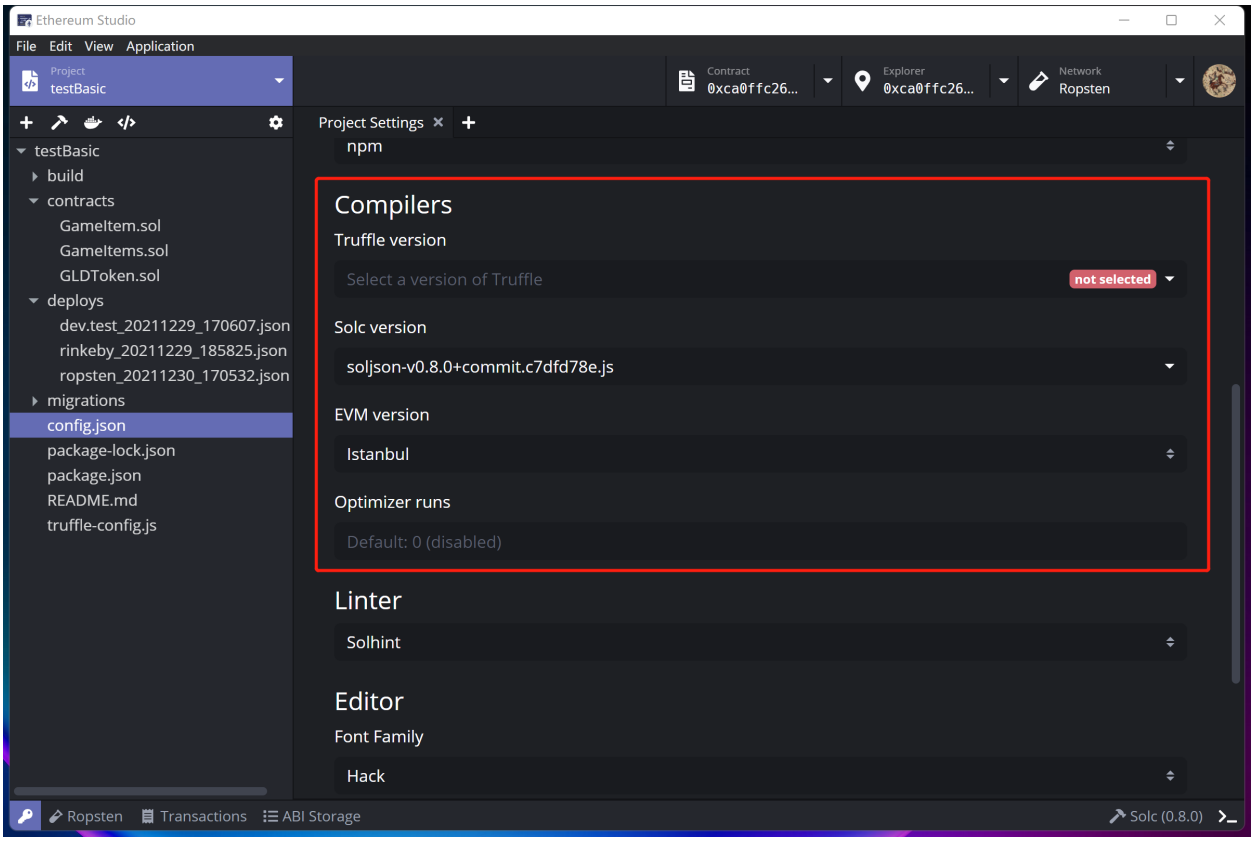
Developers can switch the framework here and then use a new framework to build or deploy with more detailed configuration in “package.json”.

Developers can change clients here and use a new client like yarn or cnpm. Please make sure that developers has installed the clients and runs directly in the command line.



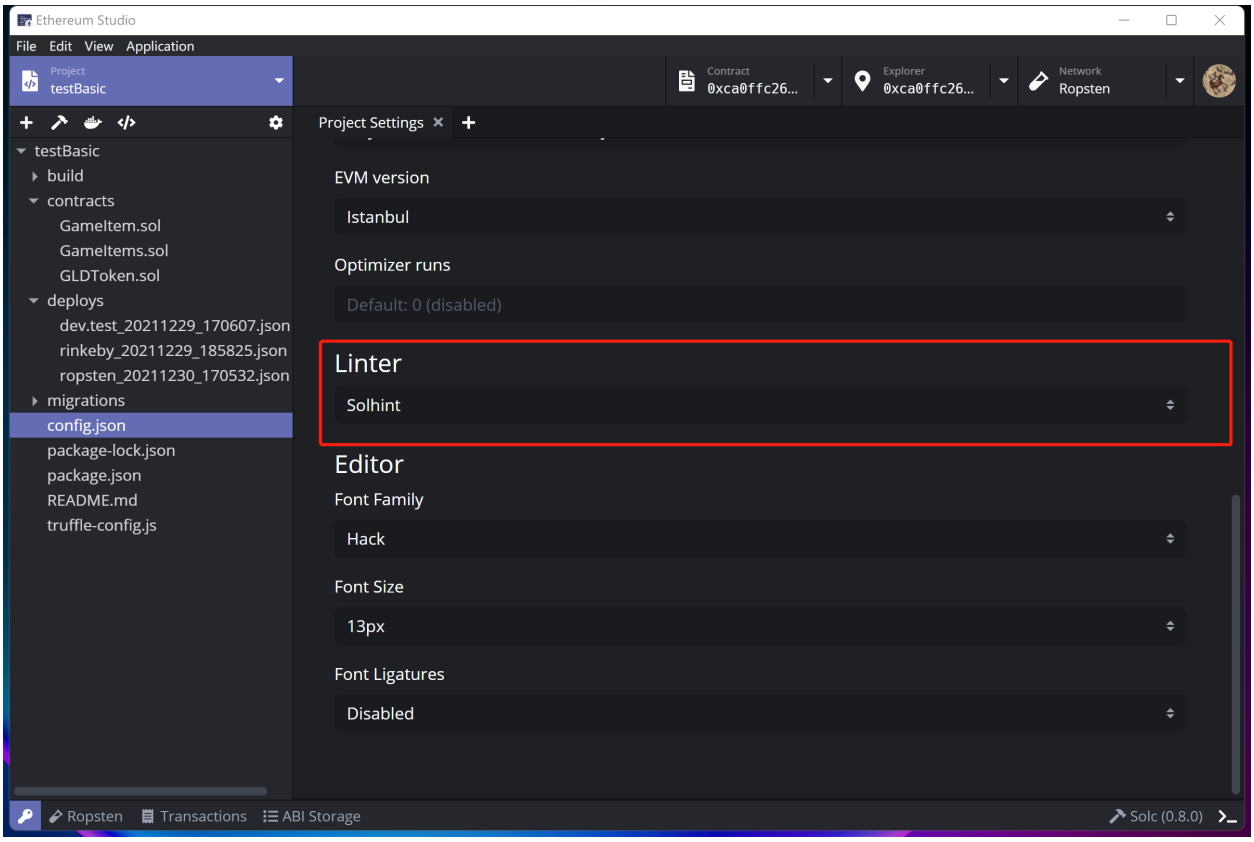
5.3.2 Compilers

Developers can select the Truffle version while the default version is settled during the creating process. The "Solc(0.x.y)" version is identical to the pragma version in the head of solidity file. Developers can change the EVM version and Optimizer directly. The Ethereum Studio disabled Optimizer???



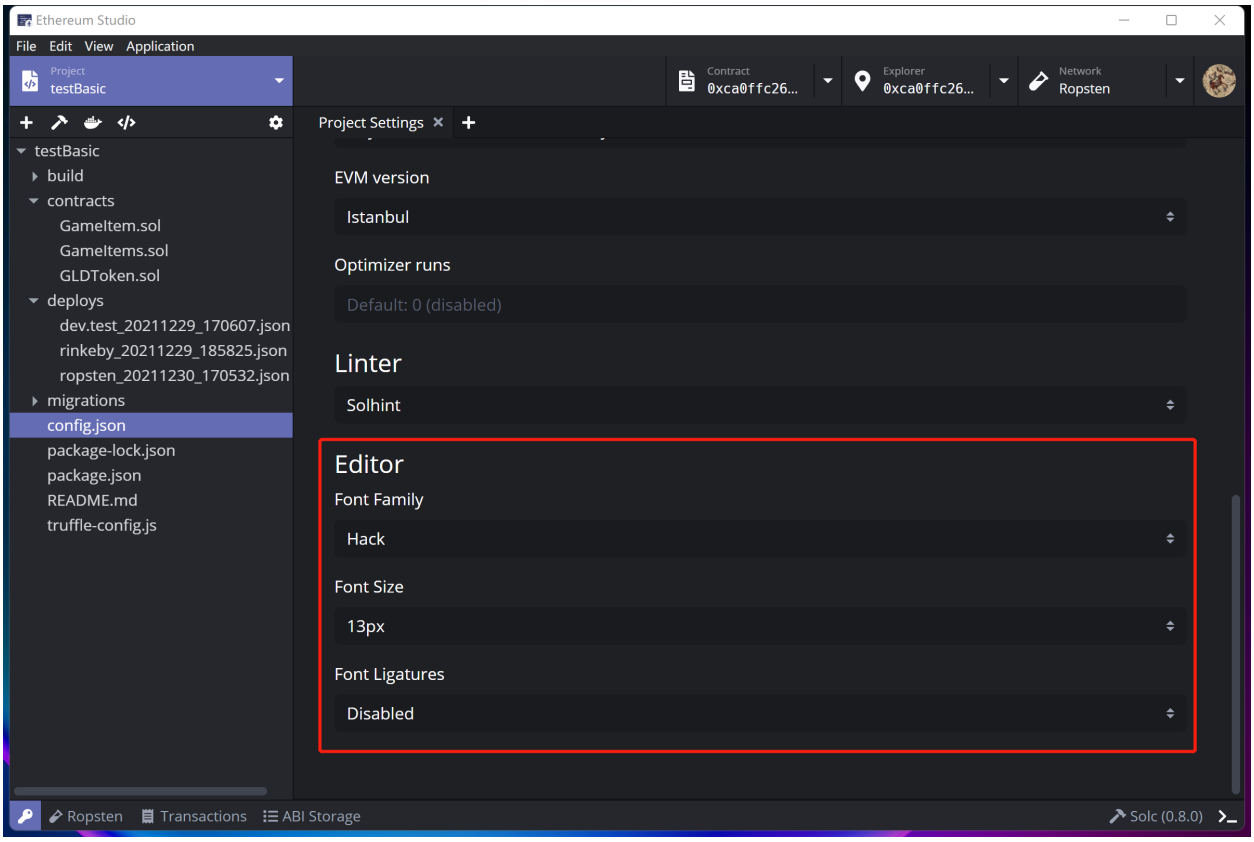
5.3.3 Linter

Linters analyze code for possible programmatic and styling errors automatically. In “Project Settings”, there are Solhint and Ethlint. Developers can choose a familiar lint to complete codes.



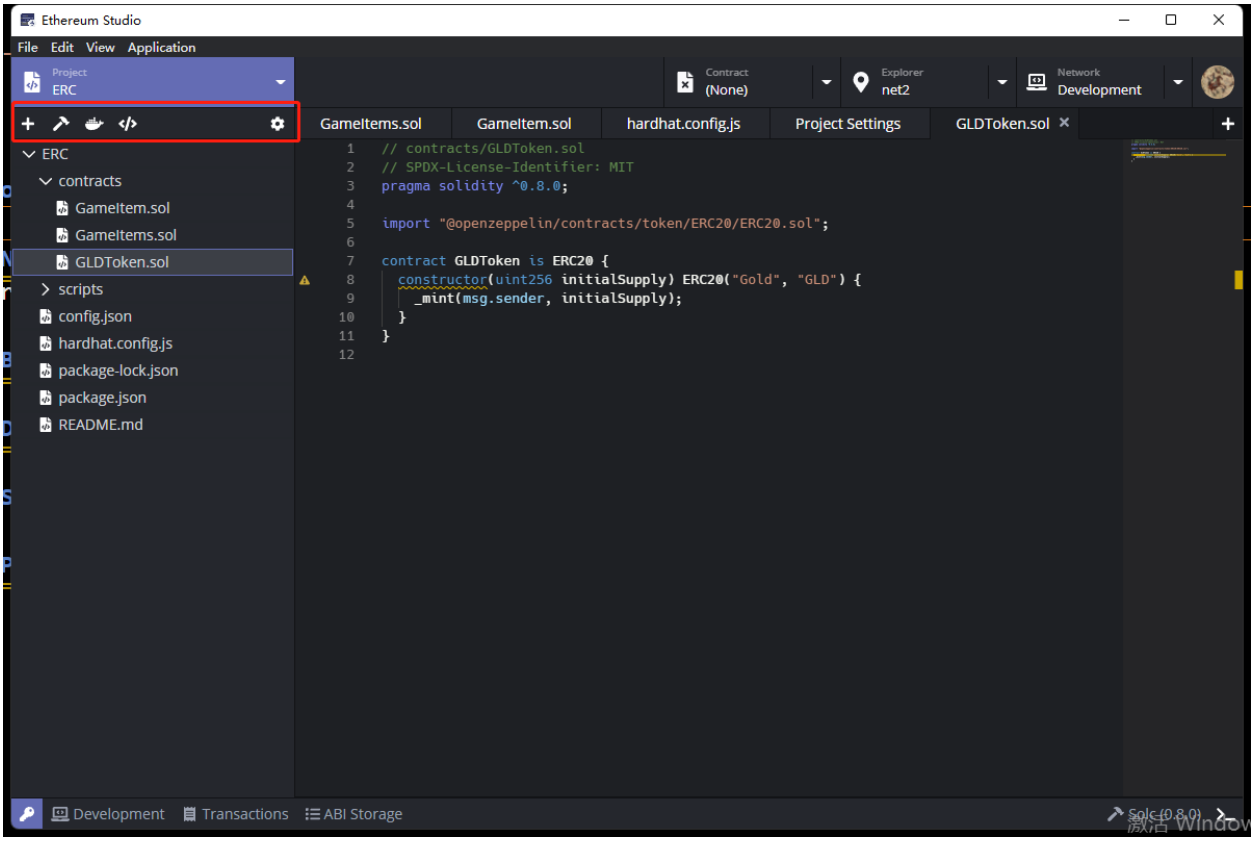
5.3.4 Editor

In “Editor”, developers can choose a font-related configuration to make code more specific and direct as desired.



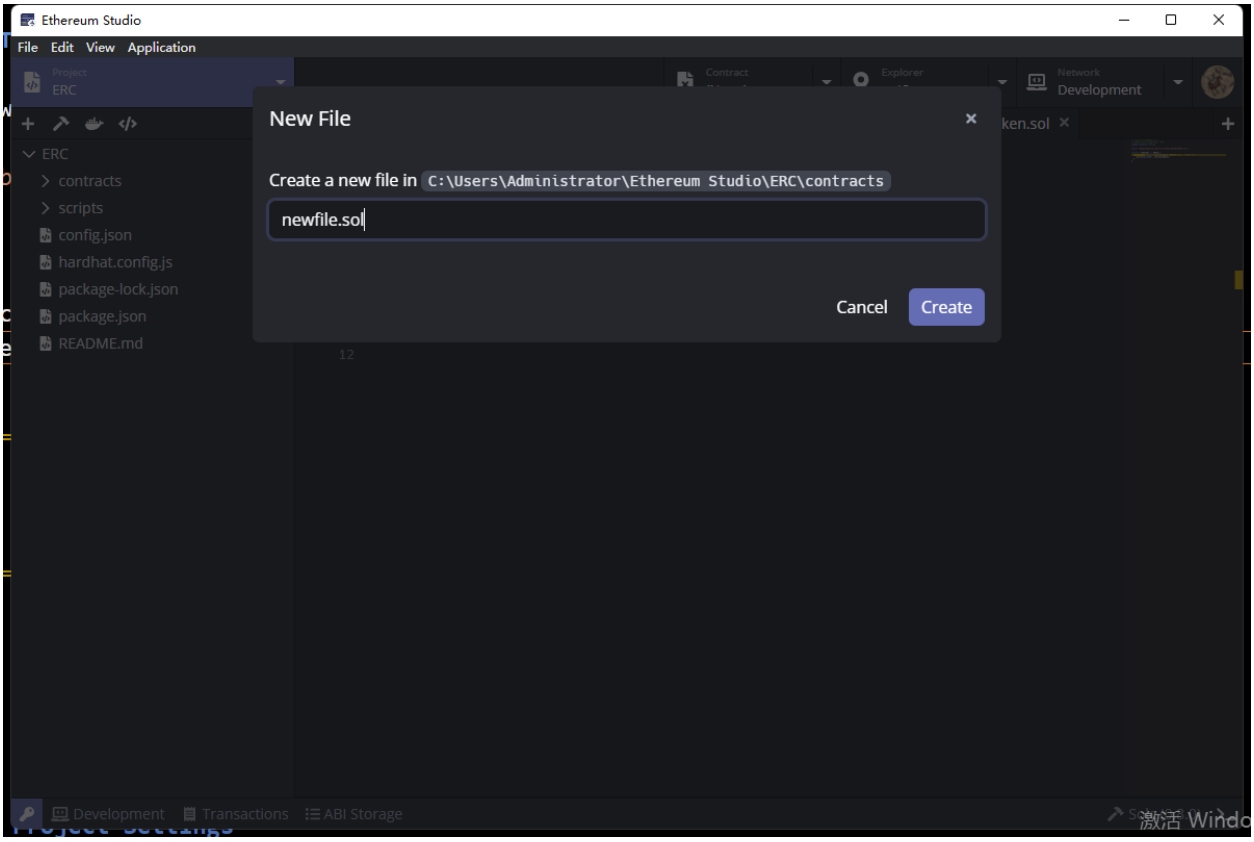
5.4 Tool Bar

Between “Project” panel and fill tree, tool bar has several quick functions for developers.



5.4.1 New File

Clicking “plus” icon named “New File”, developers can create a new file in the current path. Developers can define both name and type of the new file in the input box. Then, click the purple “Create” button and the new file will be generated successfully.



5.4.2 Build

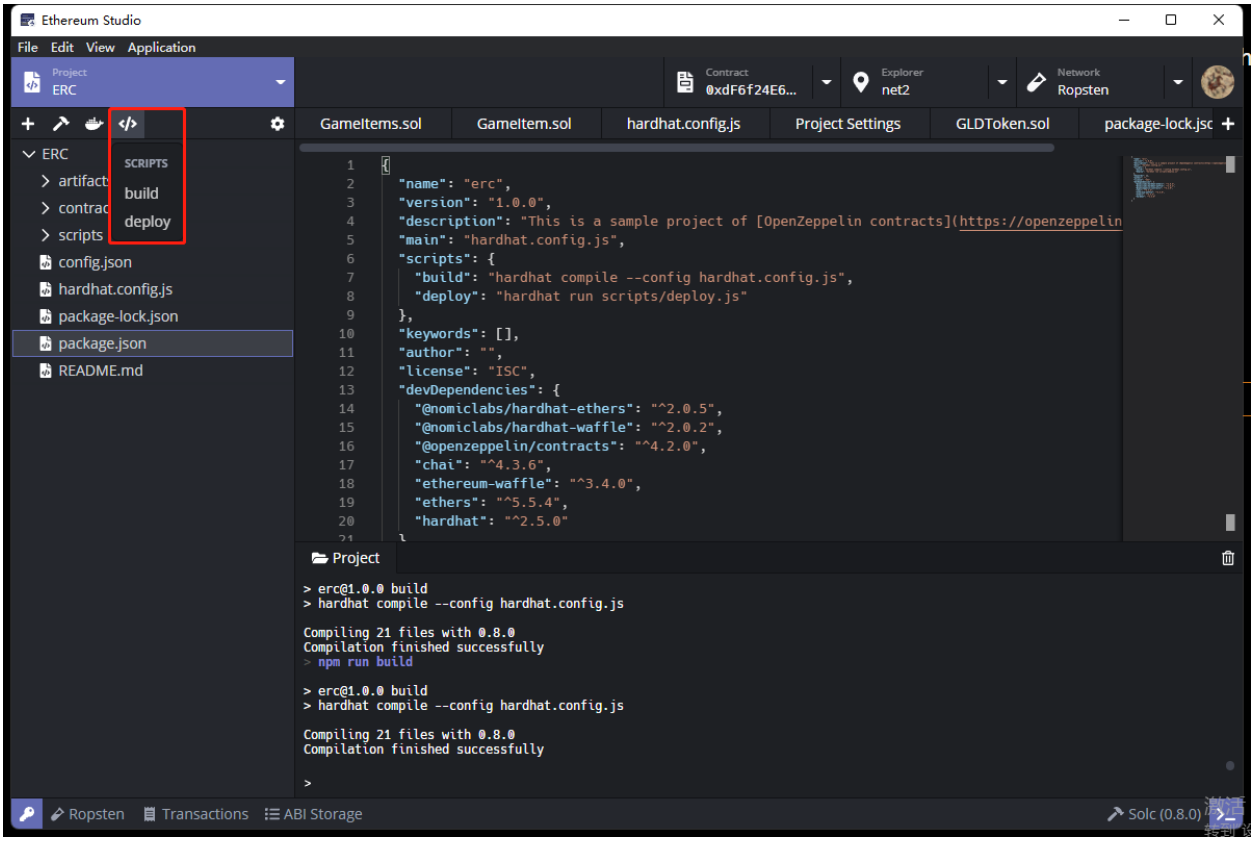
Developers can build all the contracts together quickly by clicking the “hammer” icon. The detail information for buidling can be checked in “Build” section above.

5.4.3 Deploy

Developers can deploy the contract by clicking the “docker” icon. The detail information for buidling can be checked in “Deploy” section above.

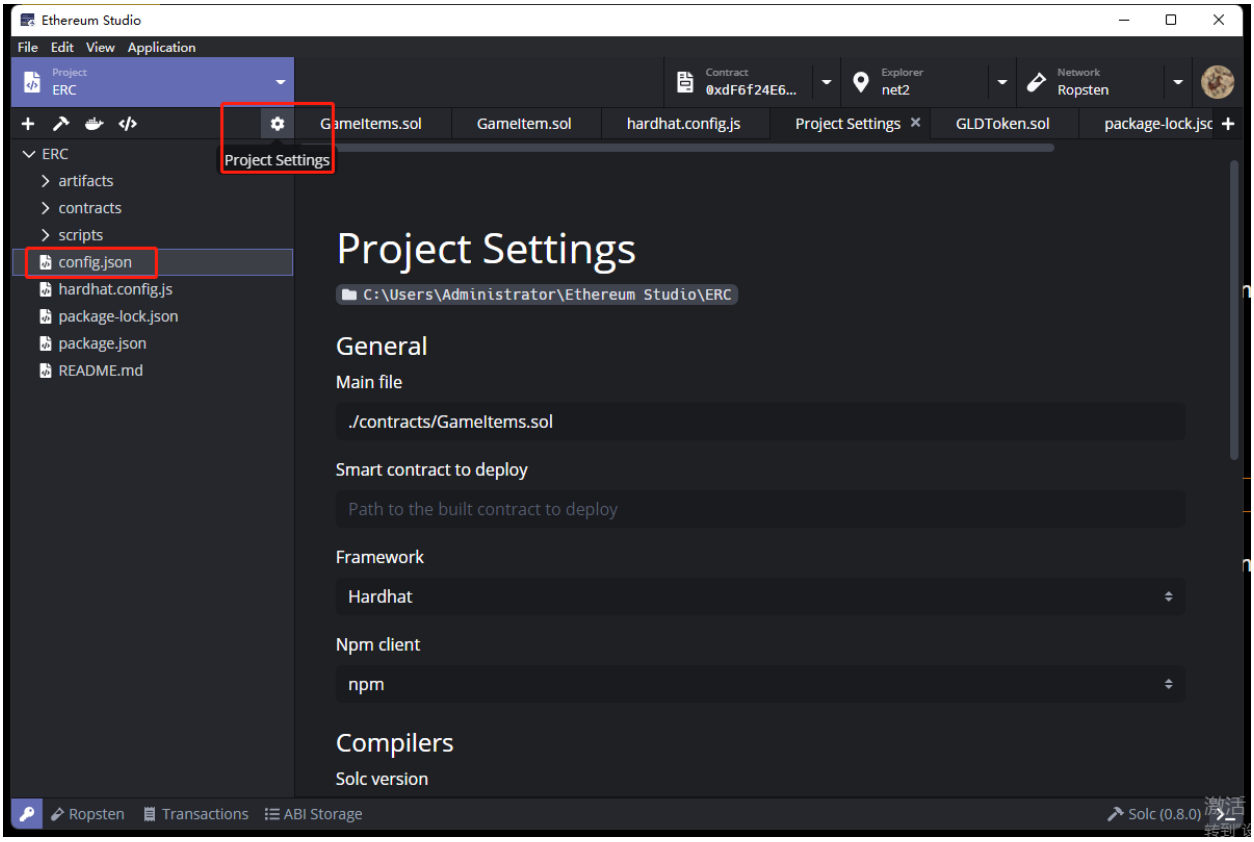
5.4.4 Script

Click the “code” icon and select “build” or “deploy” in “Script”. Then there would be corresponding command line in “script” part of “package.json” inputted in the terminal and excuted automatically.



5.4.5 Project Settings

Click the “code” icon and select “build” or “deploy” in “Script”. Then there would be corresponding command line in “script” part of “package.json” inputted in the terminal and excuted automatically.


Ethereum Studio
File Edit View Application
Project
ERC
Contract
0xdF6f24E6...
Explorer
net2
Network
Ropsten
GameItems.sol
GameItem.sol
hardhat.config.js
Project Settings
GLDToken.sol
package-lock.jsc
Project Settings
ERC
artifacts
contracts
scripts
config.json
hardhat.config.js
package-lock.json
package.json
README.md
Project Settings
C:\Users\Administrator\Ethereum Studio\ERC
General
Main file
./contracts/GameItems.sol
Smart contract to deploy
Path to the built contract to deploy
Framework
Hardhat
Npm client
npm
Compilers
Solc version
Ropsten
Transactions
ABI Storage
Solc (0.8.0)

CHAPTER

SIX

NETWORK

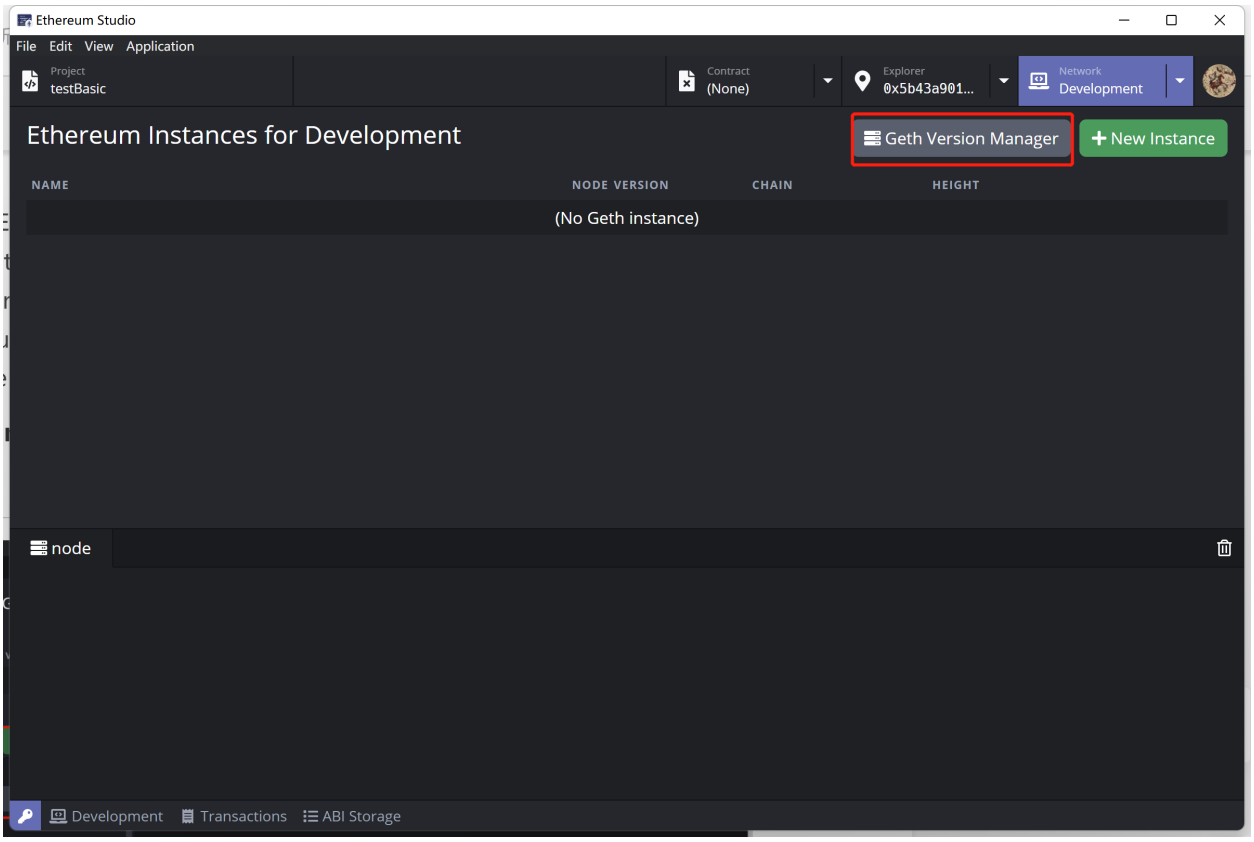
6.1 Local Development

Developers may want to run a smart contract on a local network to see how it works before deploying. In Ethereum Studio, developers can create a local blockchain instance to test smart contracts inside the IDE. This local network provides much faster develop iteration than a public testnet(for instance, you don't need to require test ETH from a testnet faucet).

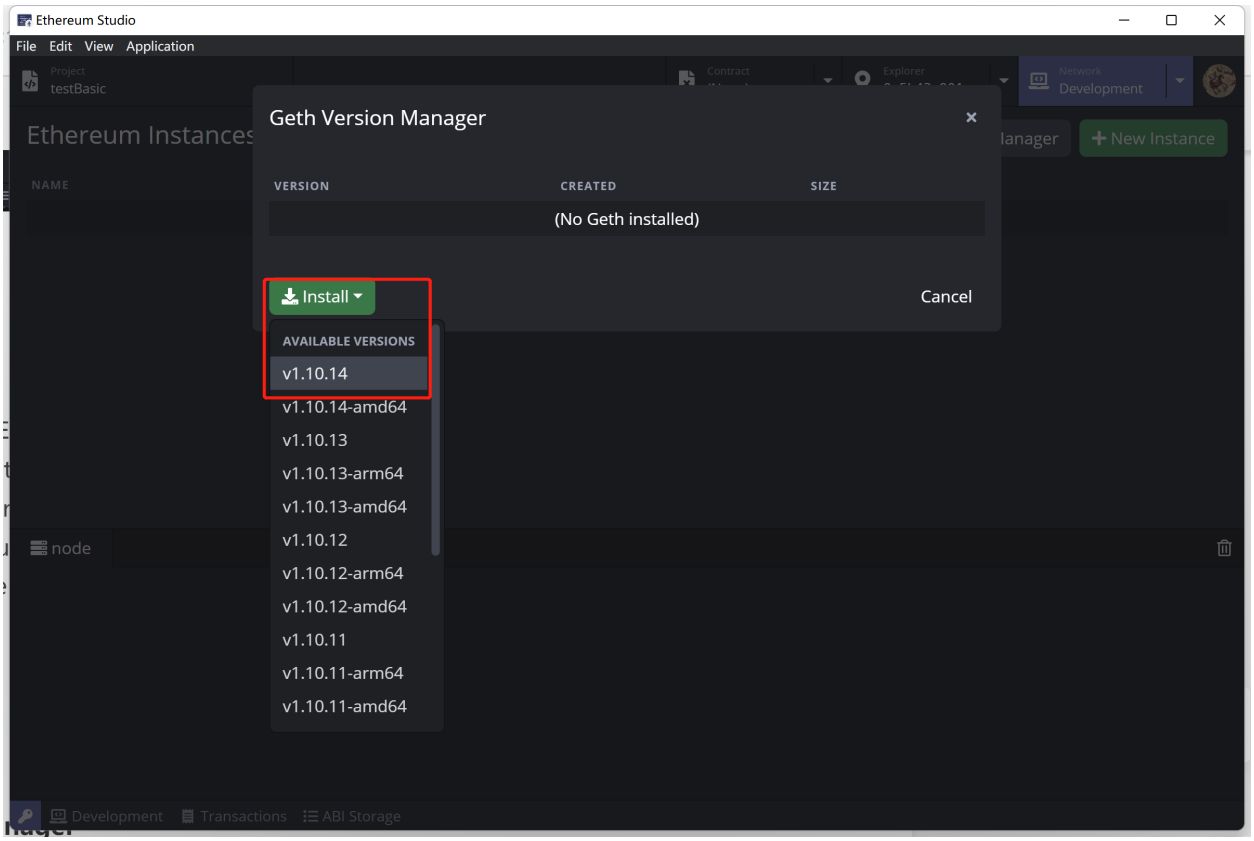
6.1.1 Geth

6.1.2 Geth Version Manager

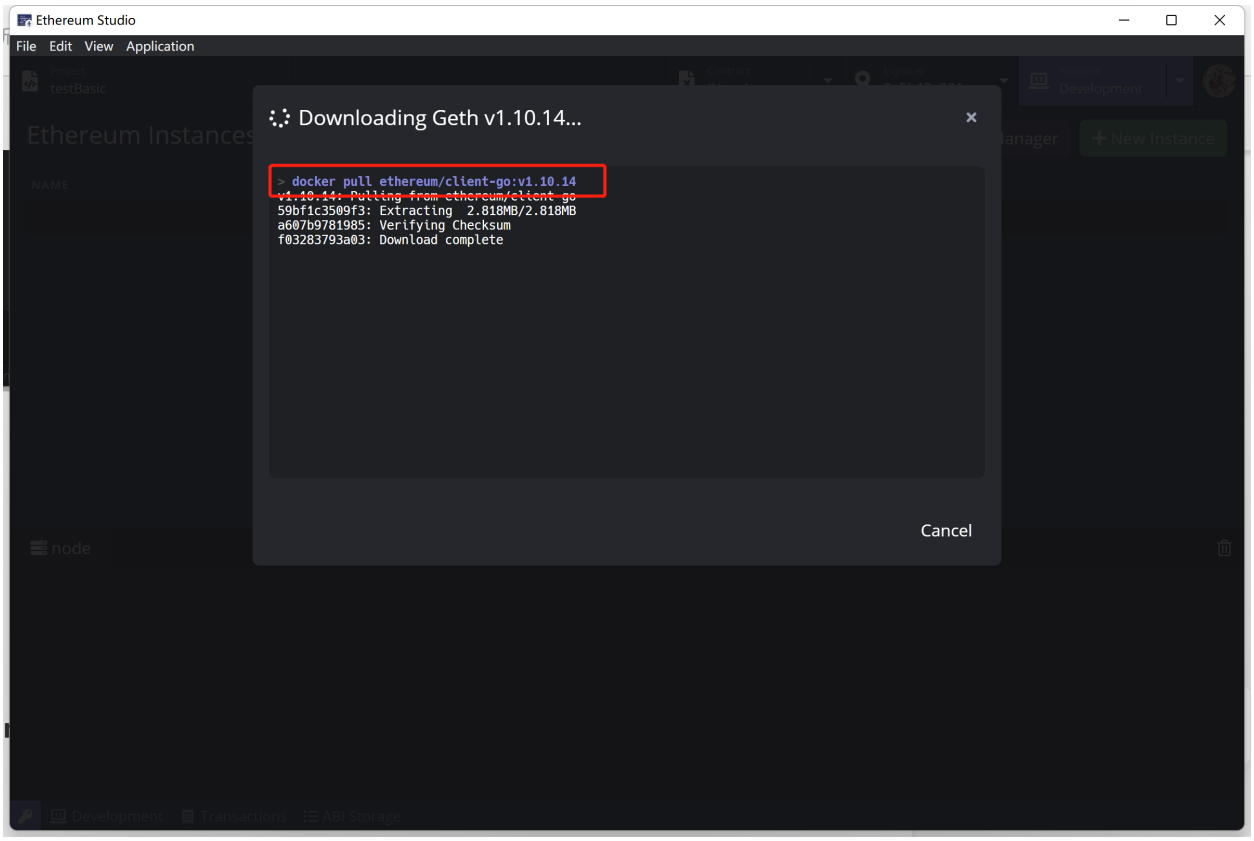
Click "Geth Version Manager" to set a specific version. Before first time launch of Ethereum Studio, developers had installed the Docker image of Geth, so there is a default version.



Developers can install a Geth version different to the default version.

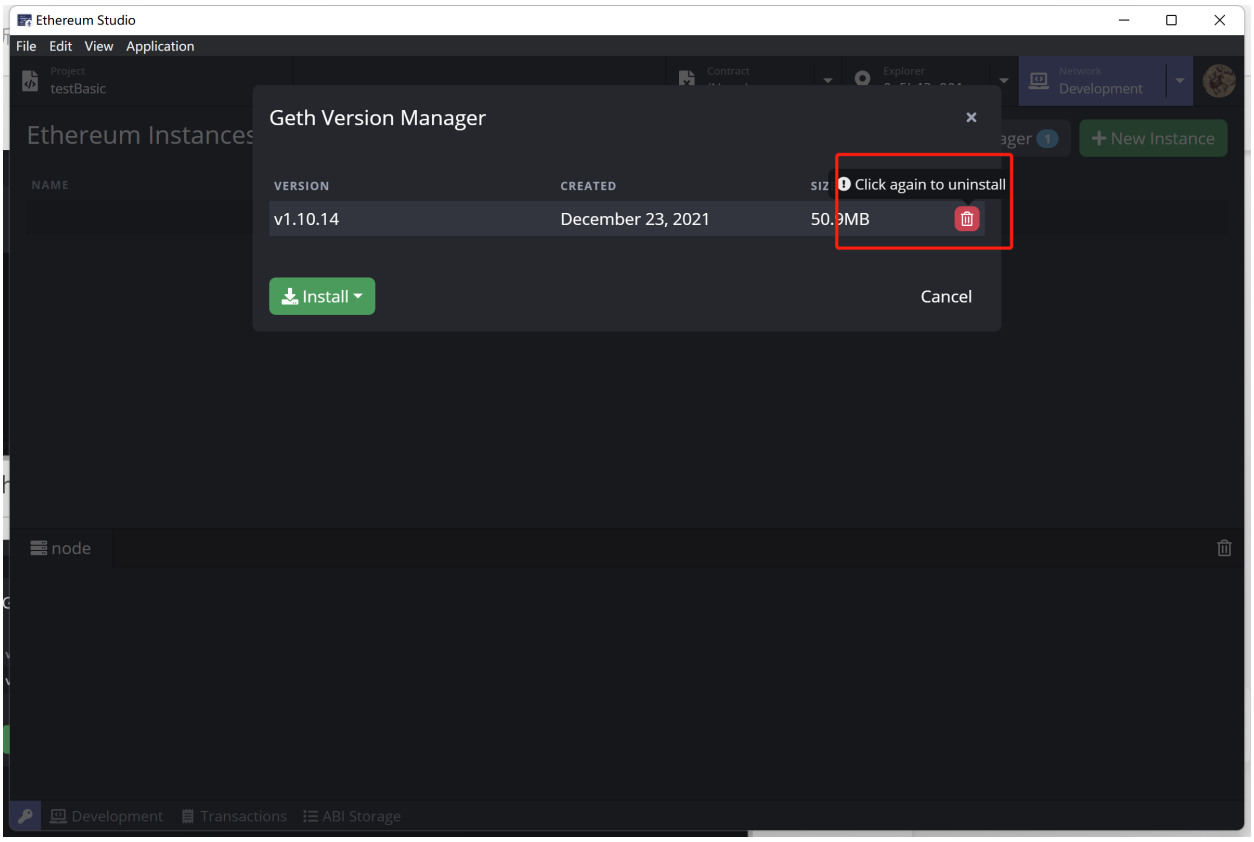


Geth is installed through Docker image. Developers has to start Docker before installing.



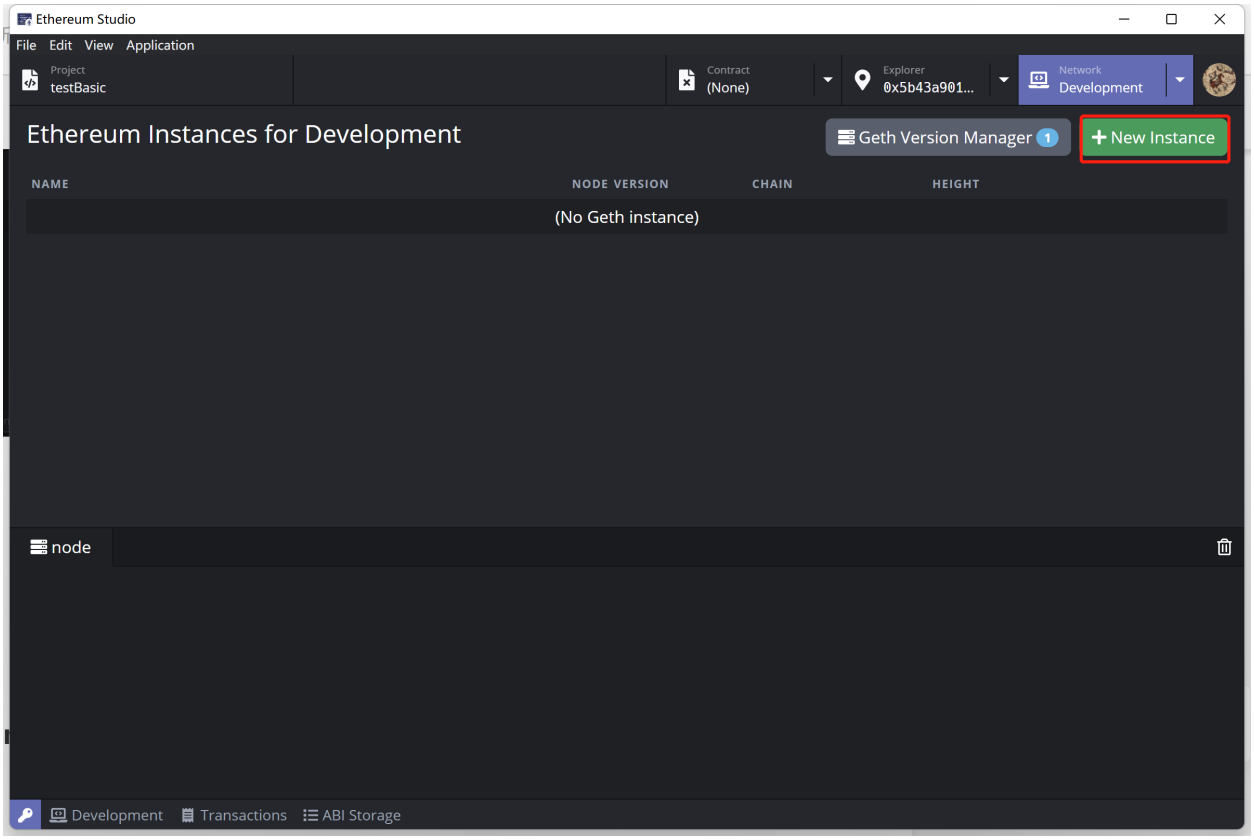
After Geth installed, developers can check Geth version in “Geth Version Manager”. There will be a blue number icon beside the “Geth Version Manager” button indicating how many versions are in the manager.

If developers want to remove the installed Geth, double-click the “trash can” icon. After the first click, the “trash can” icon will turn red, and developers can click it again to delete this Geth.

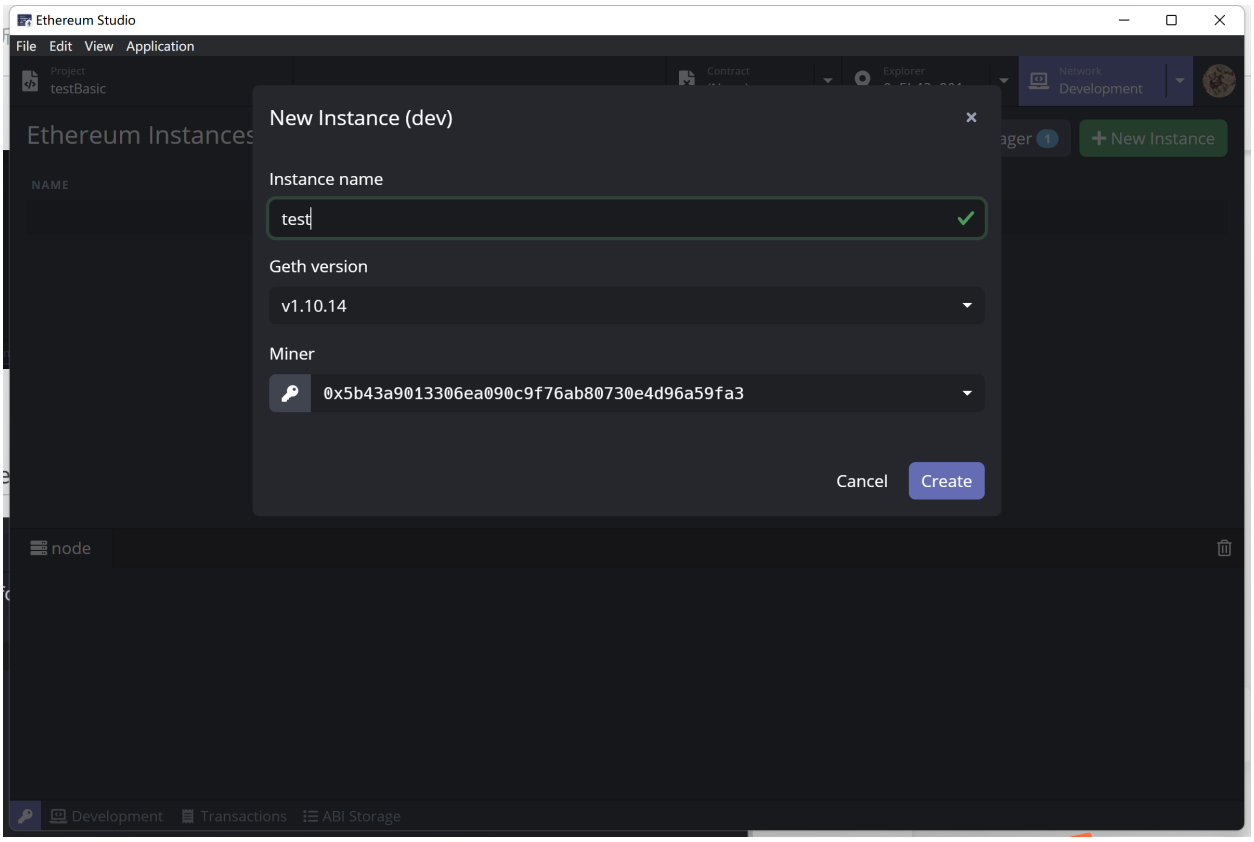


6.1.3 New Instance

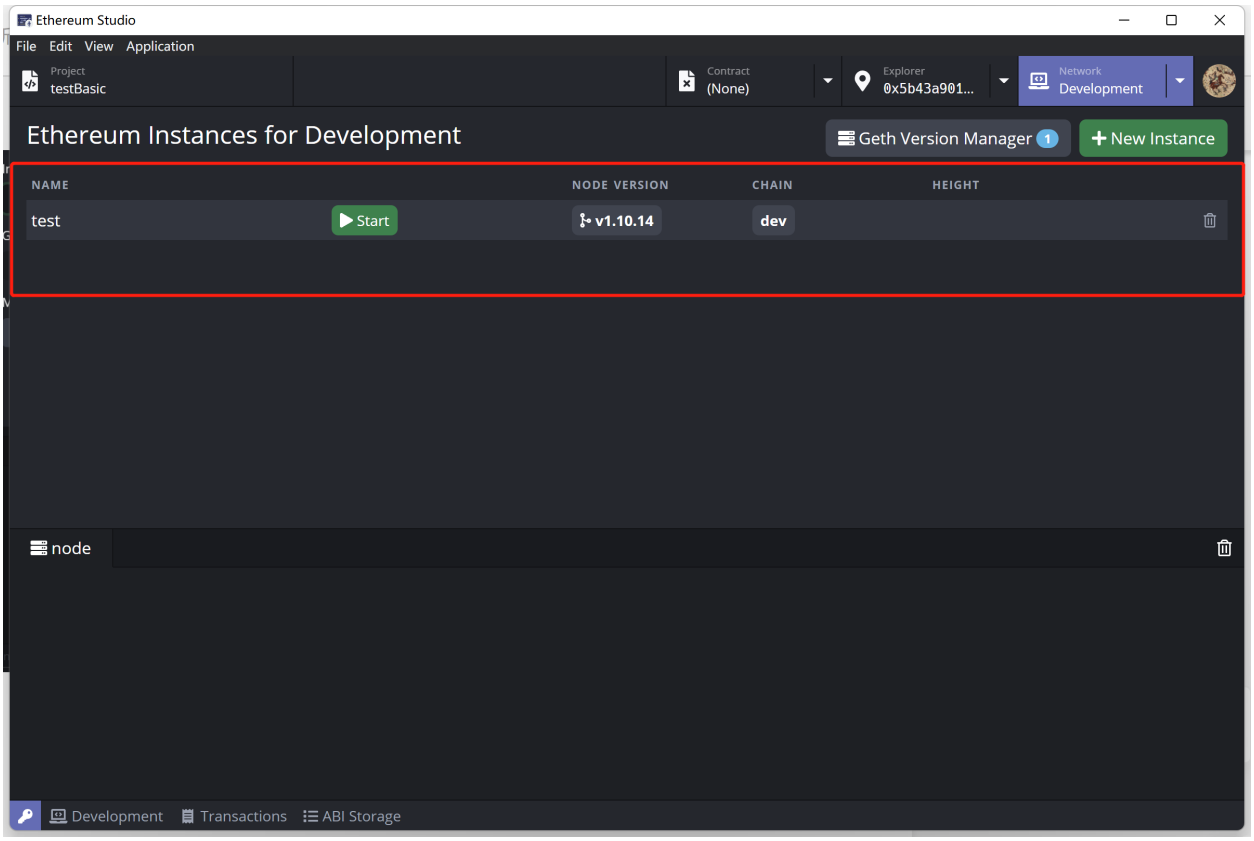
Click “New Instance” and a “New Instance (dev)” window will popup.



Developers can set “Instance name” in the window and change Geth version if there are different versions of Geth. Besides, developers can set “Miner” as the target account. Then click the “Create” button on the bottom right to make an instance.

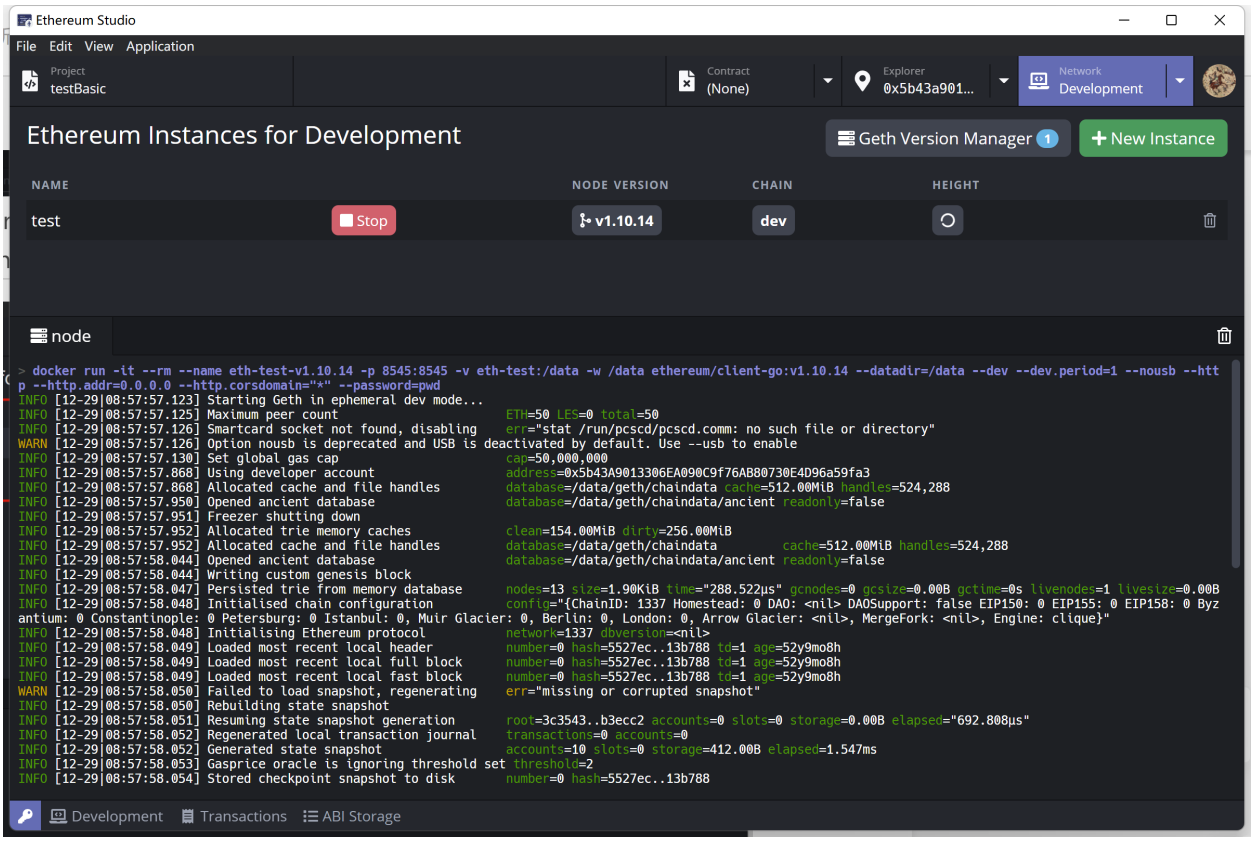


After a new instance with target Geth version created, the instance will list on the panel. Click "Start" to run the development network. Developers can create an etheruem network and connect to it locally. This local ethereum network provides default 50 ETH for user.



6.1.4 Node Panel

Click the green "Start" button on the Geth instance will start it. With information running on the node panel, the development network is prepared for developers to deploy a contract instantly. This local node cost only local ETH while users already have 50 since node running, so it is easy to test smart contracts on the private network.



6.2 Remote

6.2.1 Mainnet

A mainnet is an independent blockchain running its network with its technology and protocol. It is a live blockchain where its cryptocurrencies or tokens are in use, compared to a testnet or projects running on top of other popular networks such as Ethereum.

Ethereum Mainnet is the primary public Ethereum production blockchain, where actual-value transactions occur on the distributed ledger. Ethereum mainnet uses real ETH as currency to transfer assets and pay gas fees and tips. There are many different Ethereum testnets, and each testnet uses its own test ETH as currency, respectively. Developers may deploy and test contracts on at least one testnet before the final release on the mainnet.

6.2.2 Testnets

In addition to Mainnet, there are public testnets. These networks are used by protocol developers or smart contract developers to test both protocol upgrades and potential smart contracts in a production-like environment before deployment to Mainnet.

It's generally essential to test all smart contracts code on a testnet before deploying it to the Mainnet. Suppose developers building a dapp that integrates with existing smart contracts. In that case, most projects have copies deployed to testnets that you can interact with it.

Most testnets use a proof-of-authority consensus mechanism. This means a small number of nodes are chosen to validate transactions and create new blocks – staking their identity in the process. It's hard to incentivize mining on a

proof-of-work testnet which can leave it vulnerable.

Ropsten is a proof-of-work testnet for those running Geth, Besu and all other Ethereum clients. This means it's the best like-for-like representation of Ethereum. Ropsten started in November 2016 and it can be used with all clients.

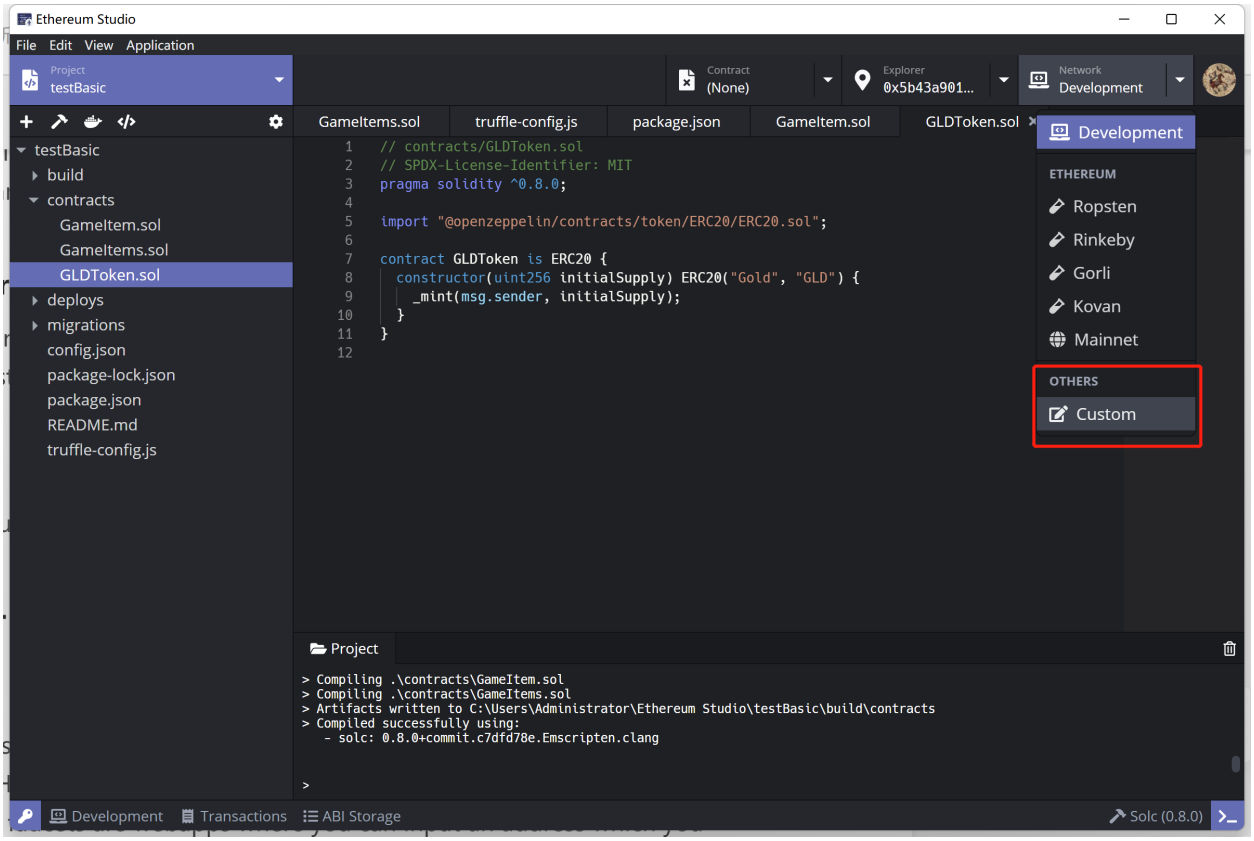
Rinkeby is a proof-of-authority(clique) testnet for those running Geth, Besu, Nethermind, and OpenEthereum client. Rinkeby started in April 2017 and is immune to spam attacks(as Trusted parties control ether supply).

Goerli is a proof-of-authority testnet that works across clients. Goerli started in November 2018. Goerli doesn't fully reproduce the current production environment as it uses PoA.

Kovan is a proof-of-authority testnet for those running OpenEthereum clients. Kovan started in March 2017 and is immune to spam attacks. Kovan doesn't fully reproduce the current production environment as it uses PoA.

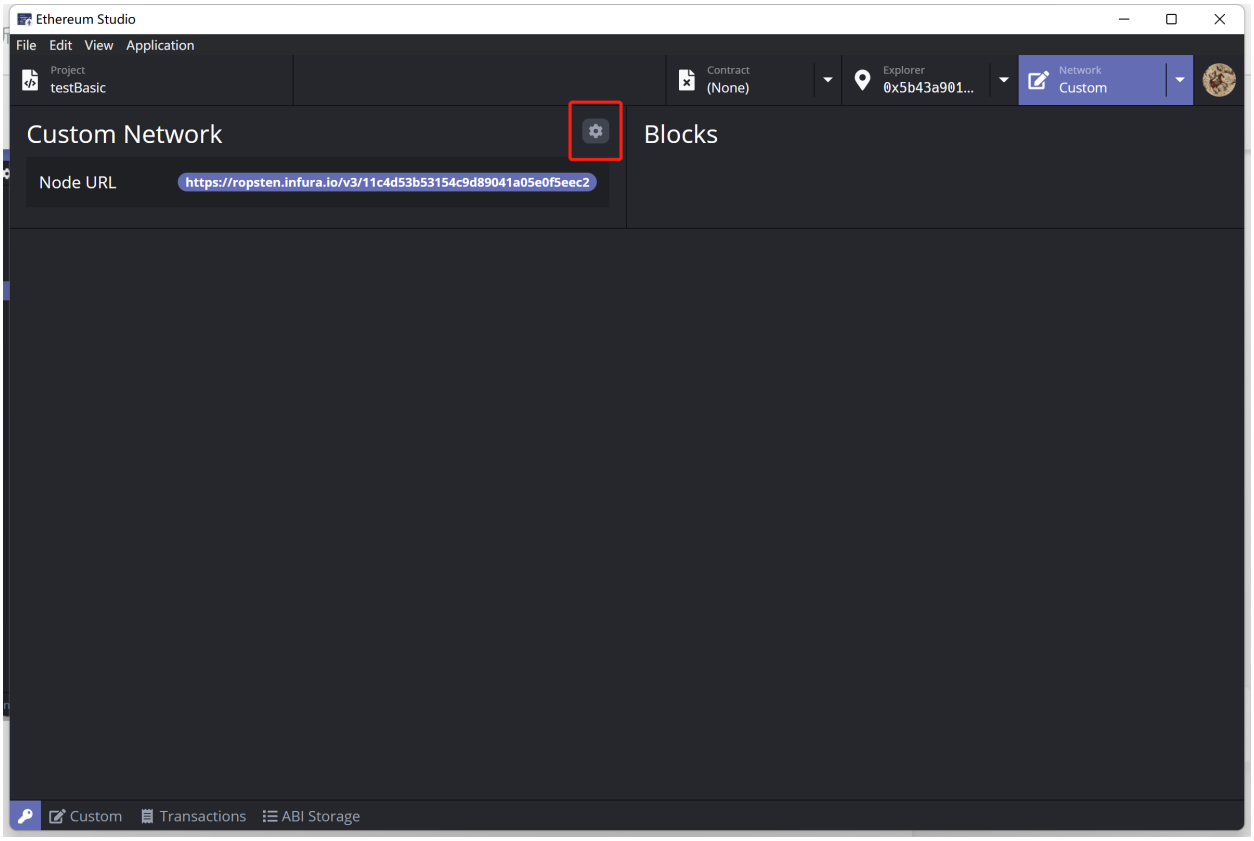
6.2.3 Custom Network

In the Ethereum network, there are private networks and public networks. The Ethereum Studio can set "Custom Network" to connect the target network. Connecting to a network, developers can join the network of other nodes instead of establishing a network by oneself. Especially, developers can use a company's network service like Infura **Link**(https://infura.io/docs).

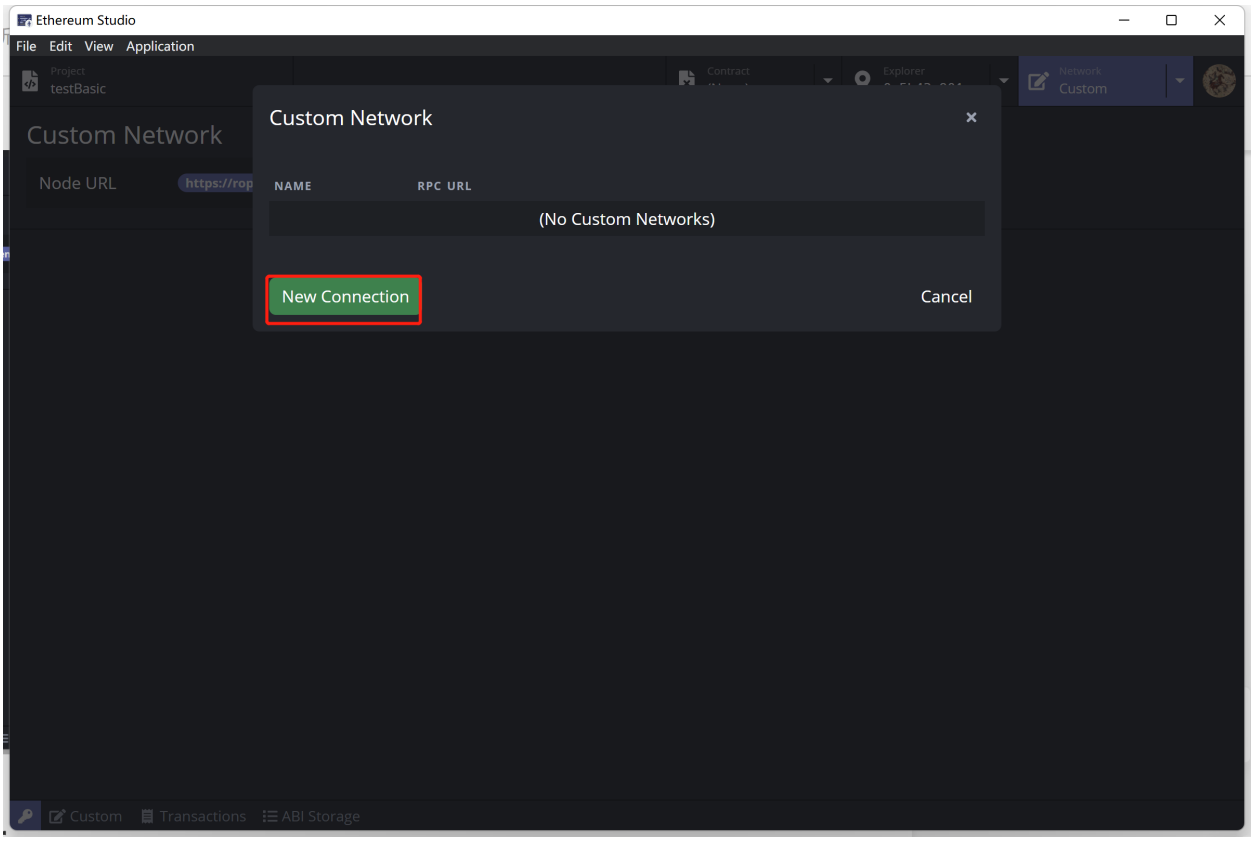


New Connection

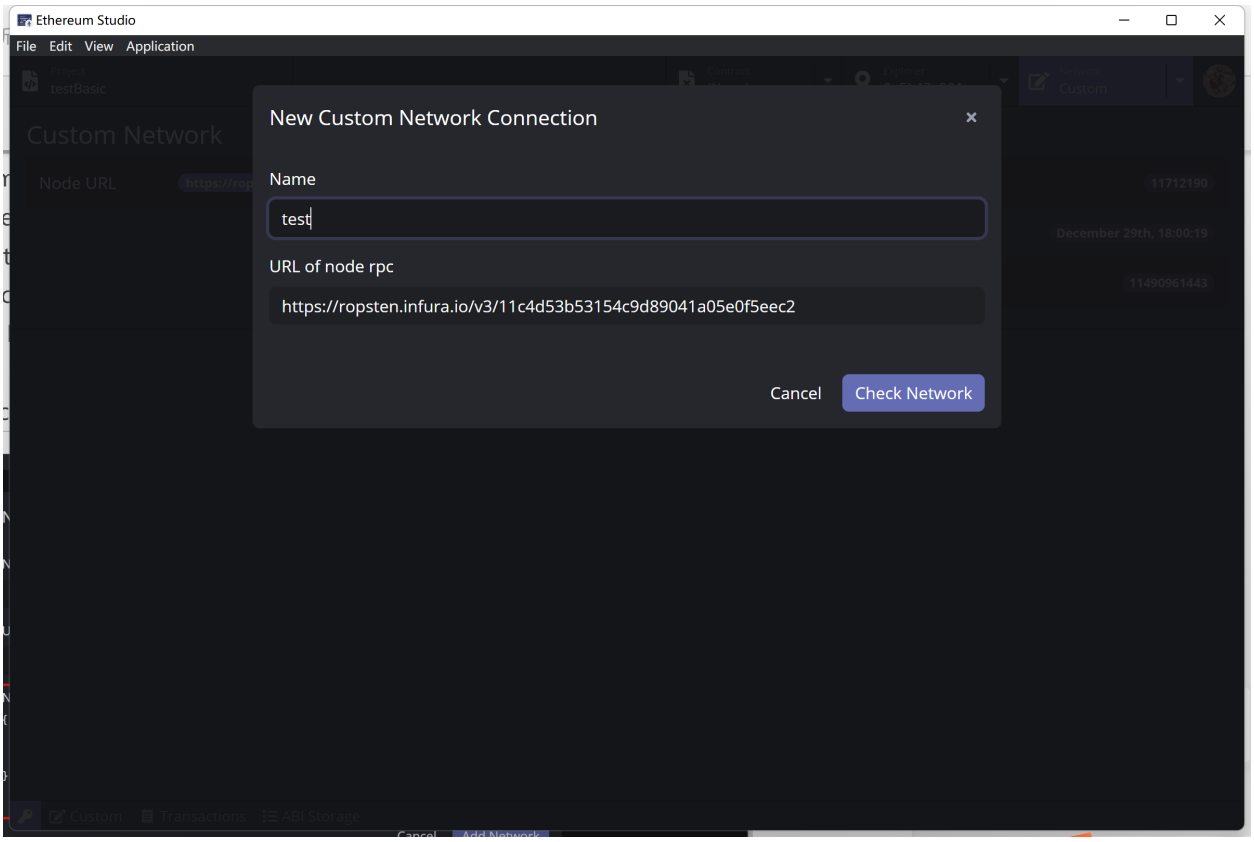
In the “Custom Network” panel, click the “gear” icon, and there will popup a window named “Custom Network”.



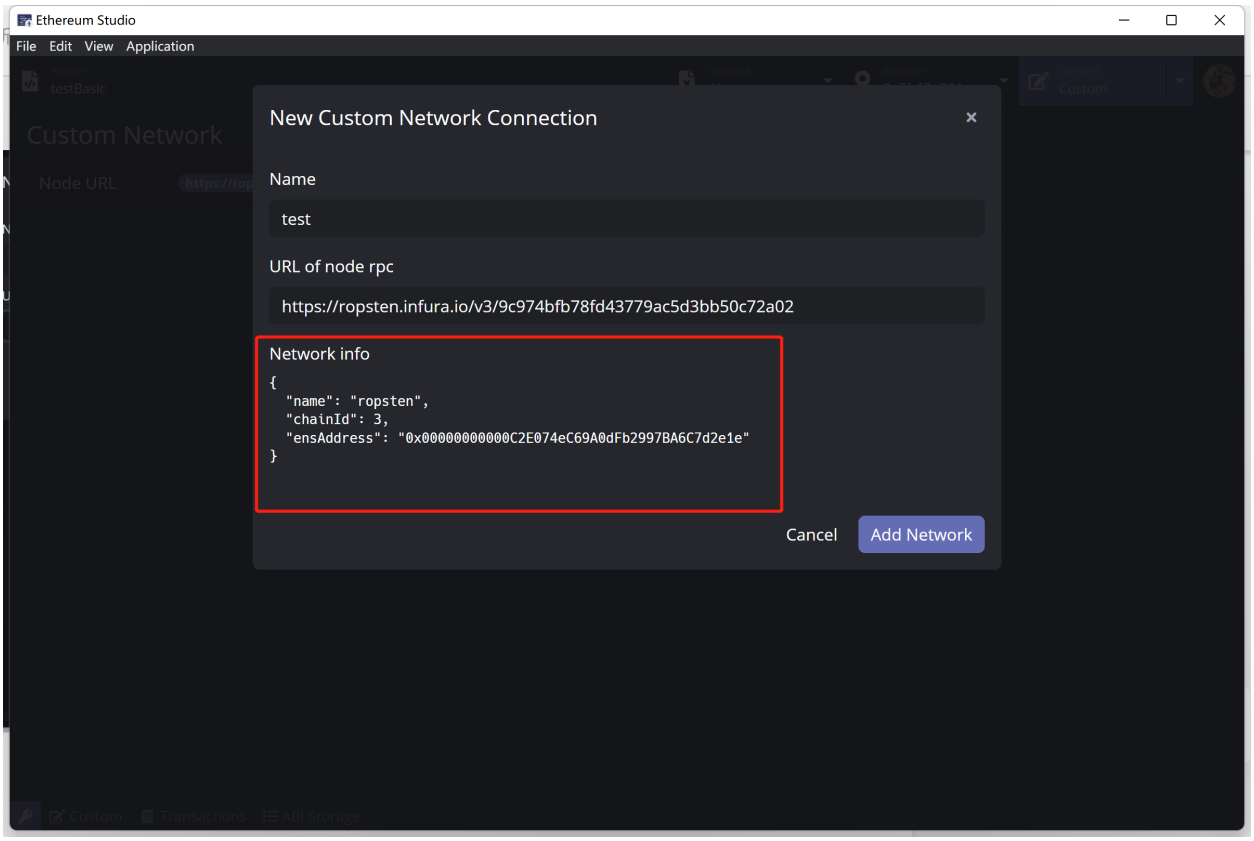
Click the “New Connection” button, and there will jump out a window for developers to add more network connections. Since each connection represents a node from one of the public or private networks, there will usually be a lot of different connections for developers to connect.



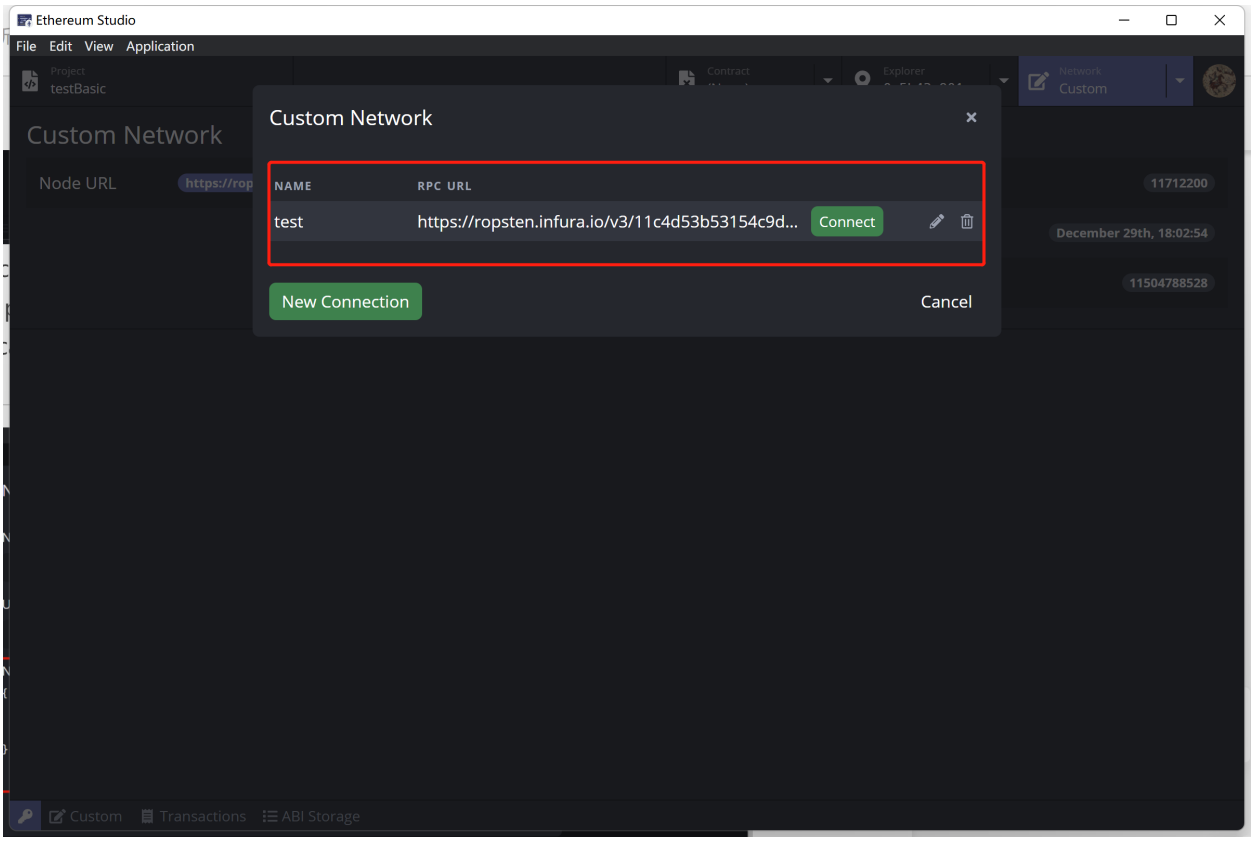
In the "New Custom Network Connection" window, developers can input the name and URL of node RPC. In this picture, there is a node of Ropsten network from Infura. Infura is a Web3 backend and Infrastructure-as-a-Service (IaaS) provider that offers blockchain developers a range of services and tools. Developers can use Infura as a fundamental infrastructure of Ethereum projects. Besides, developers can join other Geth nodes with the node parameters.

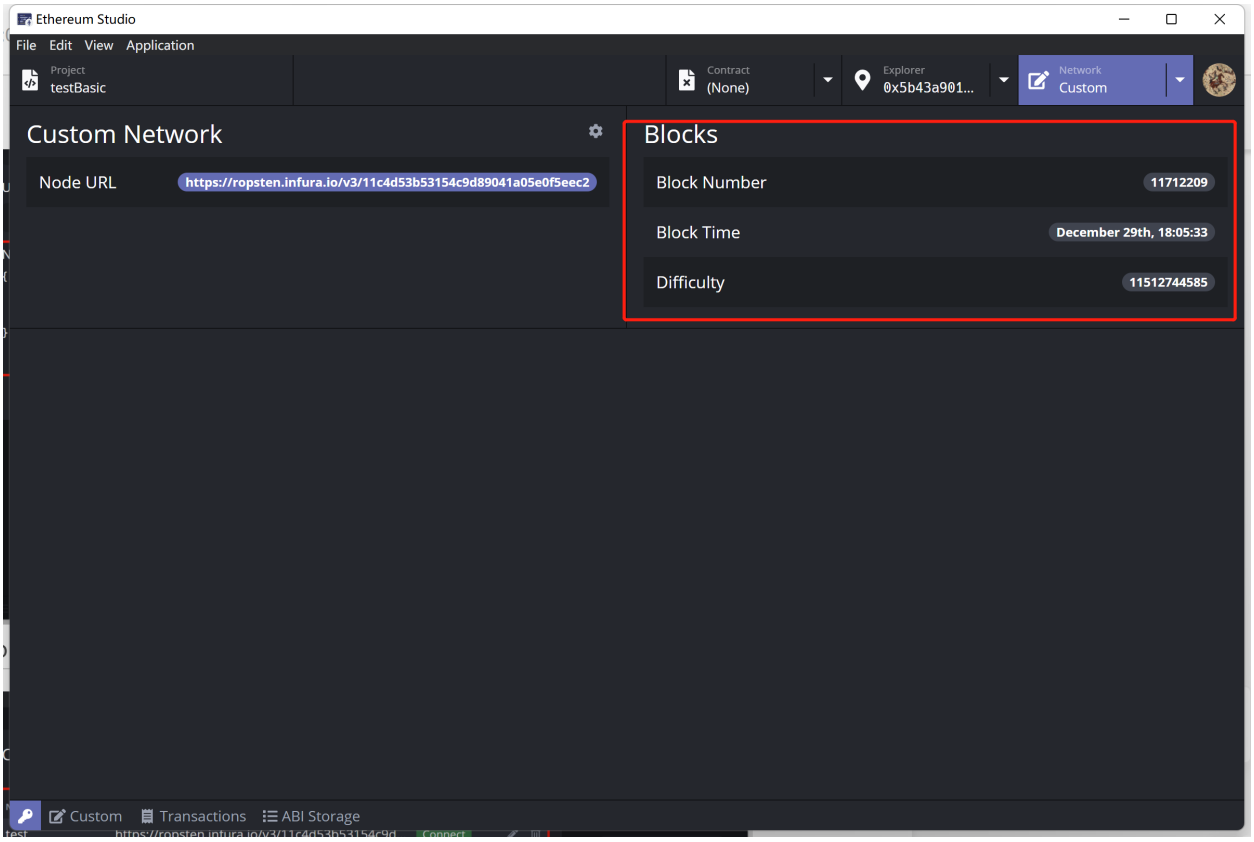


After clicking “Check Network,” a “Network info” will be added below the original panel. The panel shows detailed network information to be joined with “URL of node RPC”. Developers can check the information and join the network by clicking the “Add Network” button.



After adding a network, click the green “Connect” button, and there will be a “Blocks” panel showing the “Block Number”, “Block Time”, and “Difficulty” in real-time.


Ethereum Studio
File Edit View Application
Project
testBasic
Contract
Explorer
Network
Custom
Custom Network
Node URL
https://rop
11712200
December 29th, 18:02:54
11504788528
Custom Network
NAME
RPC URL
test
https://ropsten.infura.io/v3/11c4d53b53154c9d...
Connect
New Connection
Cancel
Custom
Transactions
ABI Storage


Ethereum Studio
File Edit View Application
Project
testBasic
Contract
(None)
Explorer
0x5b43a901...
Network
Custom
Custom Network
Node URL
https://ropsten.infura.io/v3/11c4d53b53154c9d89041a05e0f5eec2
Blocks
Block Number
11712209
Block Time
December 29th, 18:05:33
Difficulty
11512744585
Custom
Transactions
ABI Storage

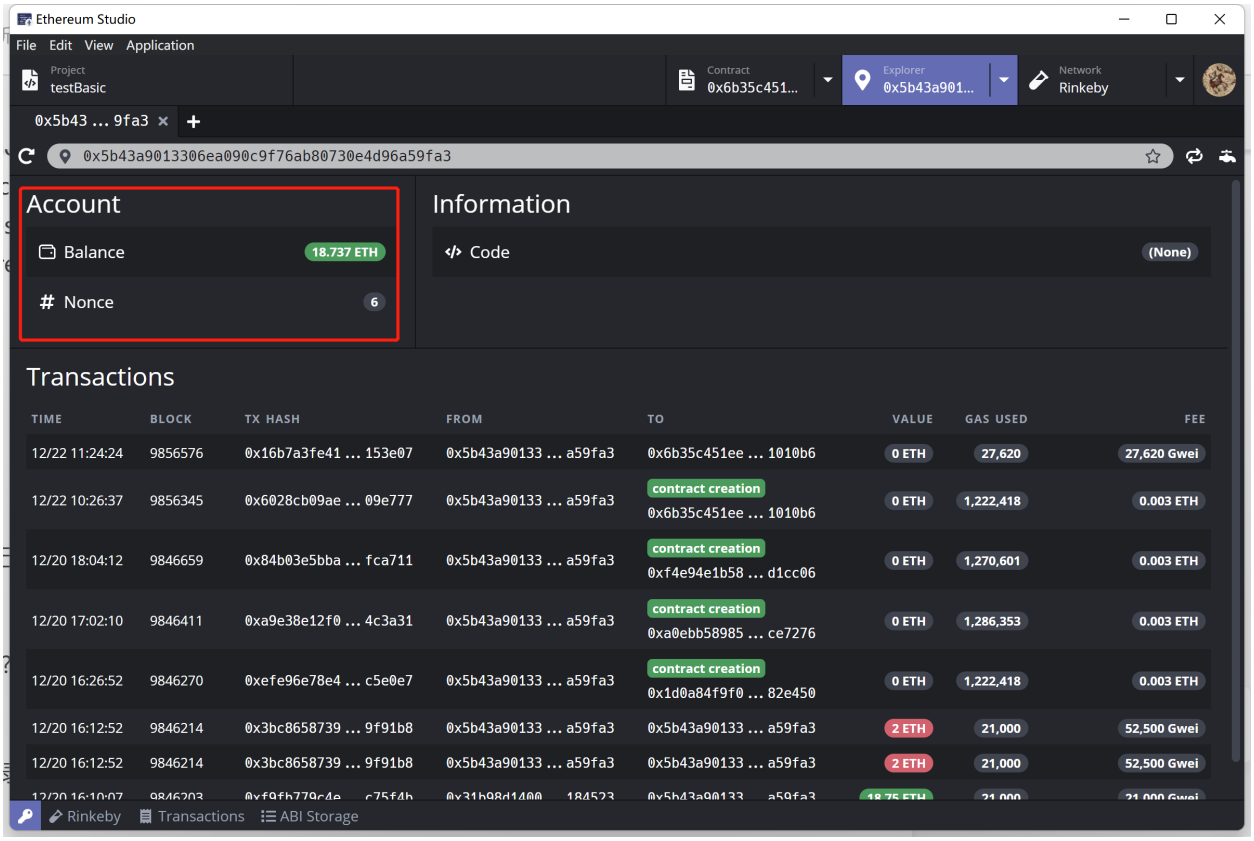
CHAPTER

SEVEN

BLOCK EXPLORER

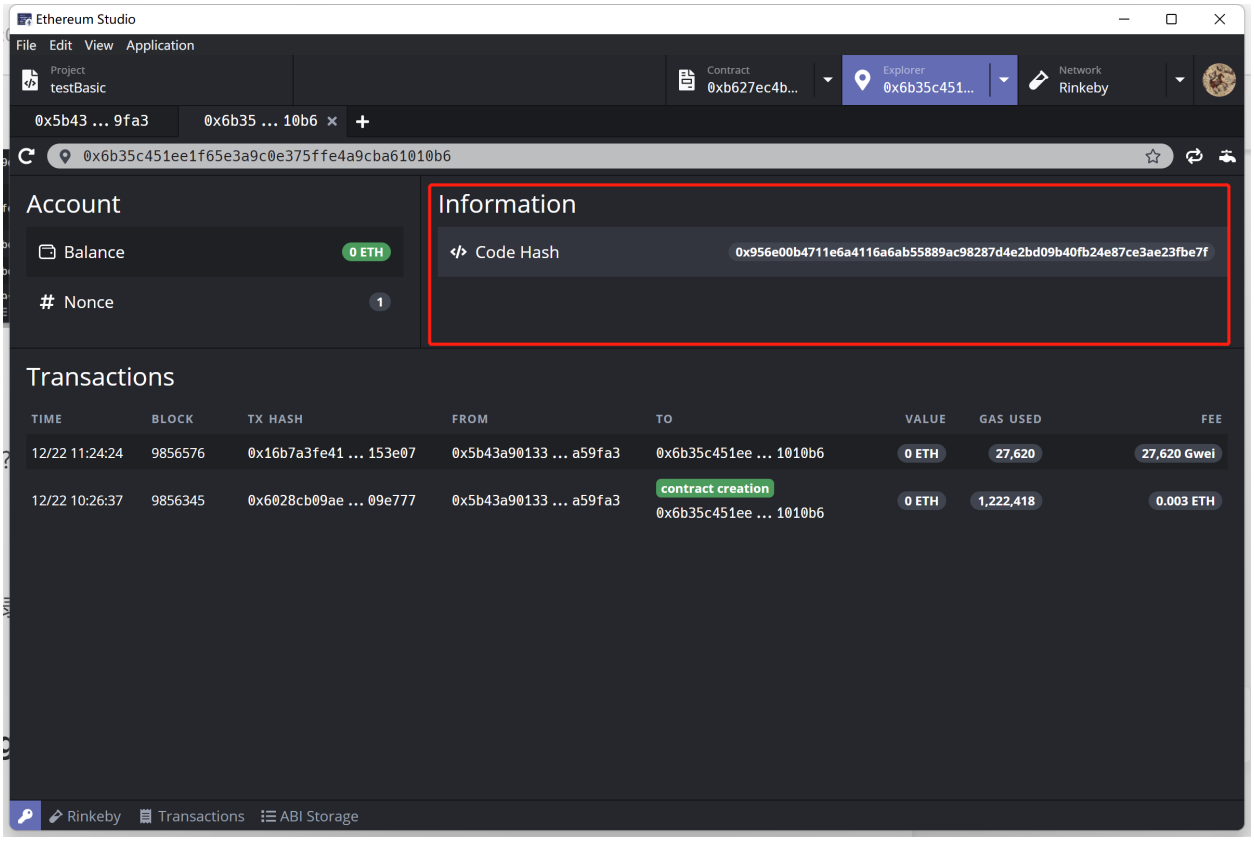
7.1 Account

In the “Account” panel, there are “Balance” and “Nounce”. “Balance” represents the ETH amount of the developers. The “Nounce” represents the transaction experience. Specifically, the nonce in the ETH wallet is a scalar value equal to the number of transactions sent from this address or the number of contract creations made by this account. Nonce can be changed manually.



7.2 Information

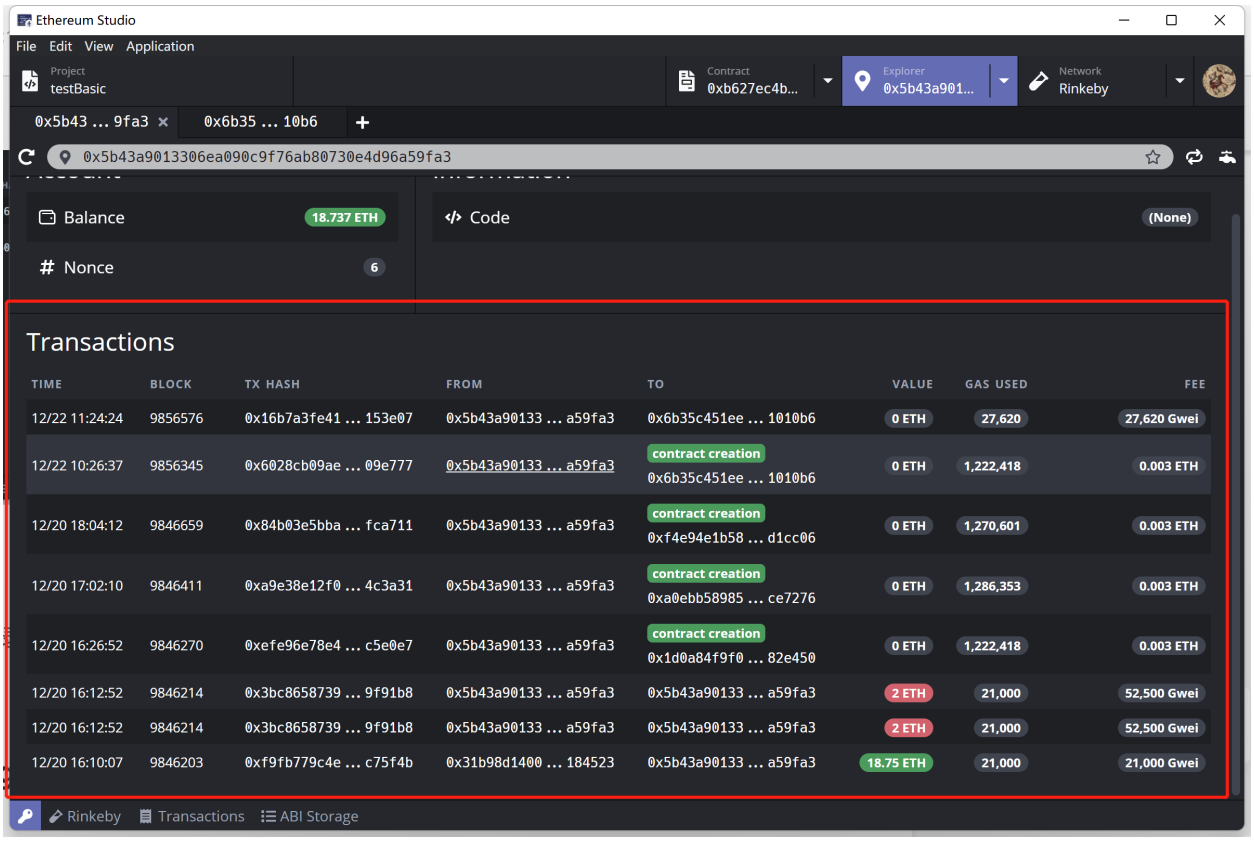
Input one of the contract address in the search box. After pressing down the "Enter" button on the keyboard, the detailed information is shown on the "Information" panel. There is the "Code Hash" of the contract in the search box in the picture.



7.3 Transactions

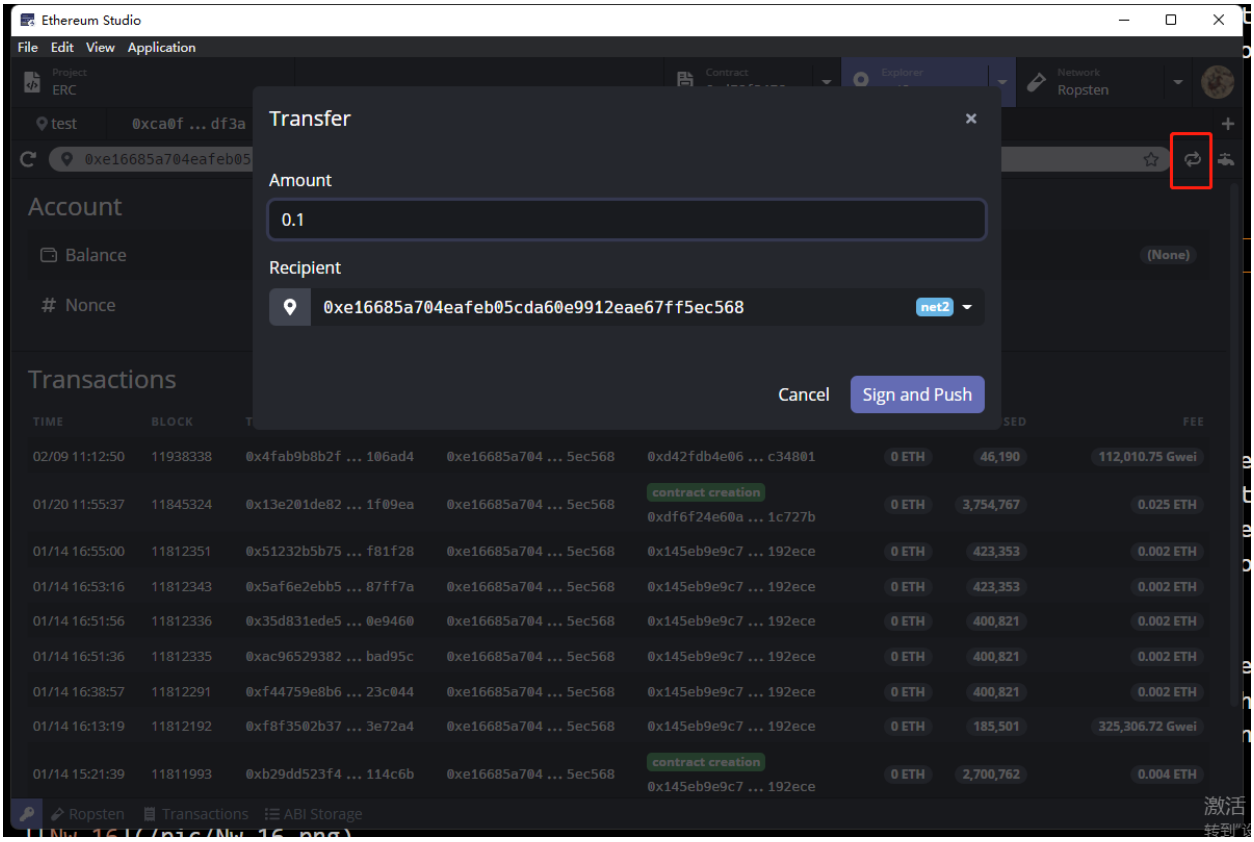
In the "Transactions" panel, there is specific information of each transaction on the address, including "Time", "Block-height", "Transaction Hash", "Owner Address", "Receiver Address", "Value" and so on.

Developers can check the colour of the transaction value to know the input or the output of ETH since the input is green and the output is red. Besides, there is detail information of the receiver address reminding the action status. Developers can click the link of address to show the address in Block Explorer to check the history information.



7.4 Transfer

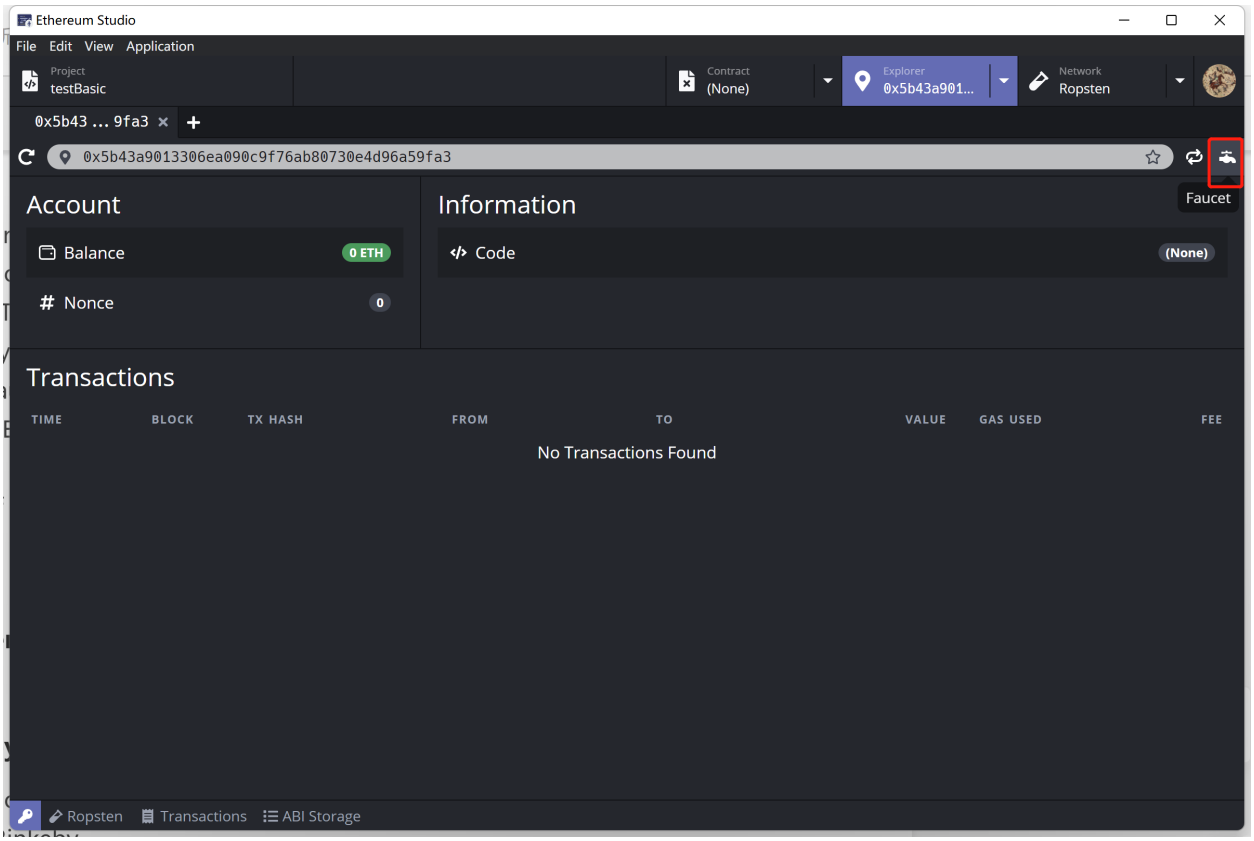
On the right of the address search bar, the “arrow cycle” icon provides transfer function for developers send ETH between address. Click it and a “Transfer” window will popup to let developers to input ETH amount and receiver address for a quick token preparation between address.



7.5 Faucet

There are currencies on the Ethereum network such as ETH and test ETH. Unlike ETH in the mainnet, test ETH has no real value. Therefore, there are no markets for testnet ETH. Most people get testnet ETH from faucets. Faucets are web apps where developers can input an address and the requested test ETH will be sent automatically.

First, choose one of the Ethereum testnets. Then set the account of Block Explorer. There is a “faucet” icon at the right end of the address search bar. Click the icon and it will turn to the faucet page of the corresponding testnet. The Ethereum Studio only supports faucets of Ropsten, Rinkeby and Kovan. Developers can check the Goerli faucet link by oneself.



7.5.1 Ropsten Faucet

Click the "Faucet" icon and the page will jump to the Ropsten Faucet. Developers can copy the wallet address and paste it into the box. Then developers clicks the button "Send me test Ether" and wait for a few minutes. There will be several test ETH of Ropensten testnet on balance.

7.5.2 Rinkeby Faucet

Following faucet instruction via inputting specific Twitter or Facebook message links, developers can get test ETH on Rinkeby. Developers can change the amount of test ETH with different lengths of time.

7.5.3 Kovan Faucet

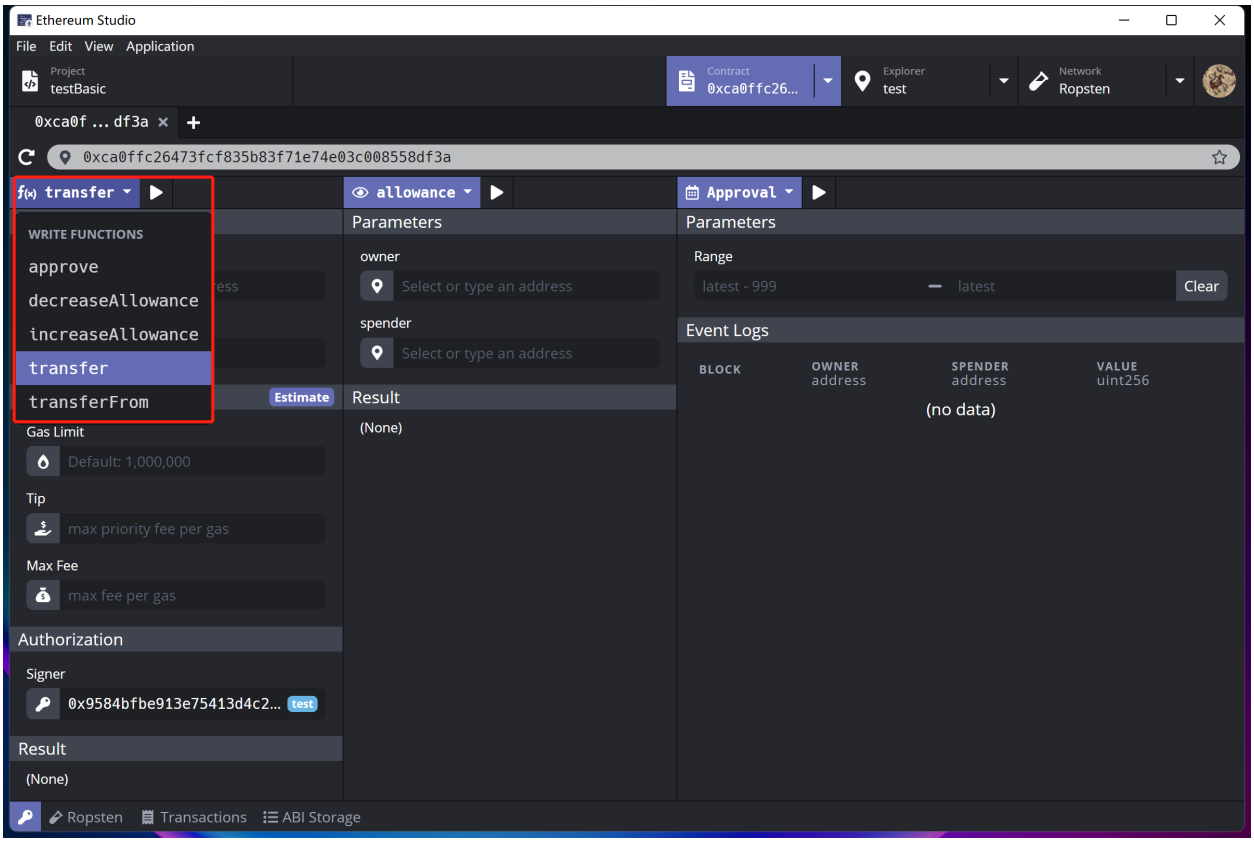
???

CHAPTER

EIGHT

CONTRACT

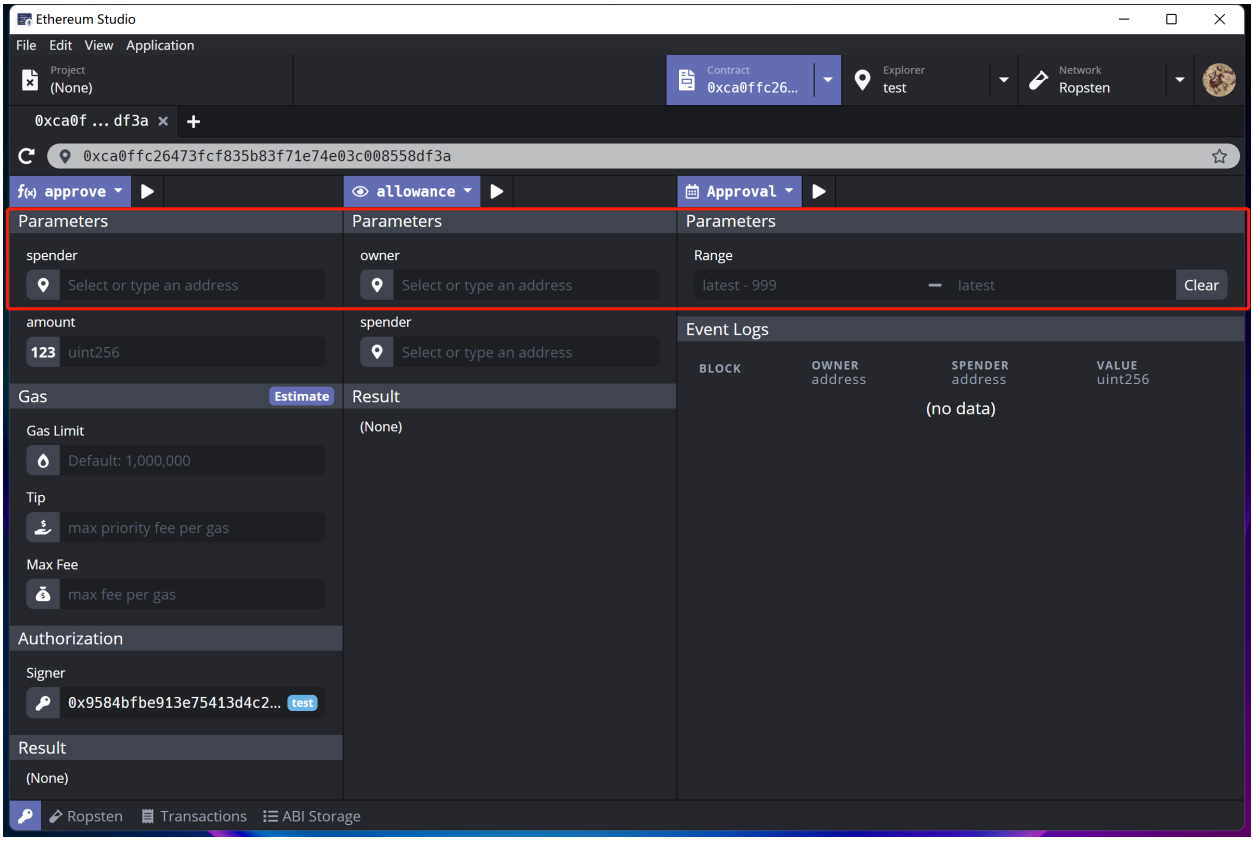
In “Contract” panel, there are three different panels from left to right including “Write Functions”, “View Functions” and “Events” respectively. Developers can interact with “Write Functions”, mainly calling functions with assets and check the address status with “View Functions”. Besides, developers can set parameters and check related events in “Events”.



8.1 Write Functions

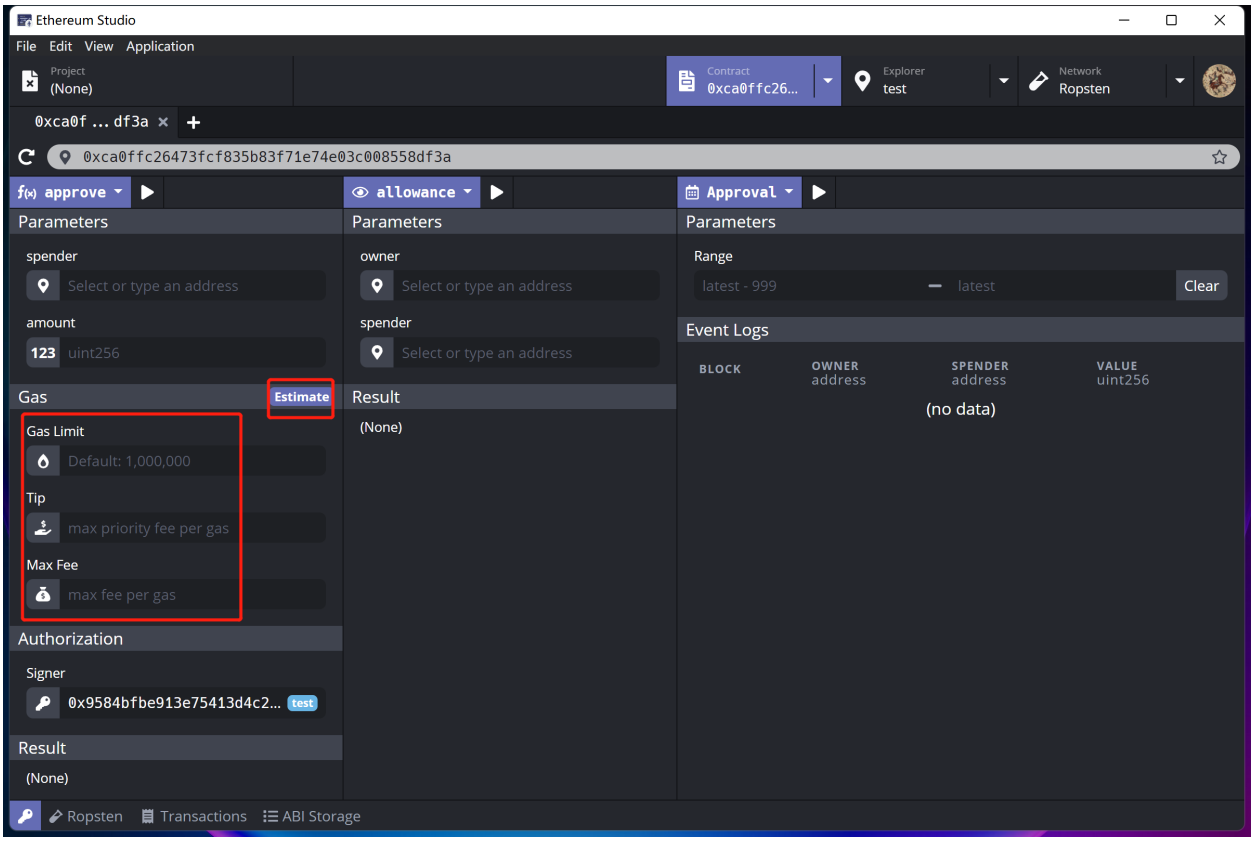
8.1.1 Parameters

In “Parameters”, developers mannually set function required parameters. Usually, there are owner, receiver, amount and so on for different functions. Please remember the token unit here is “wei”.



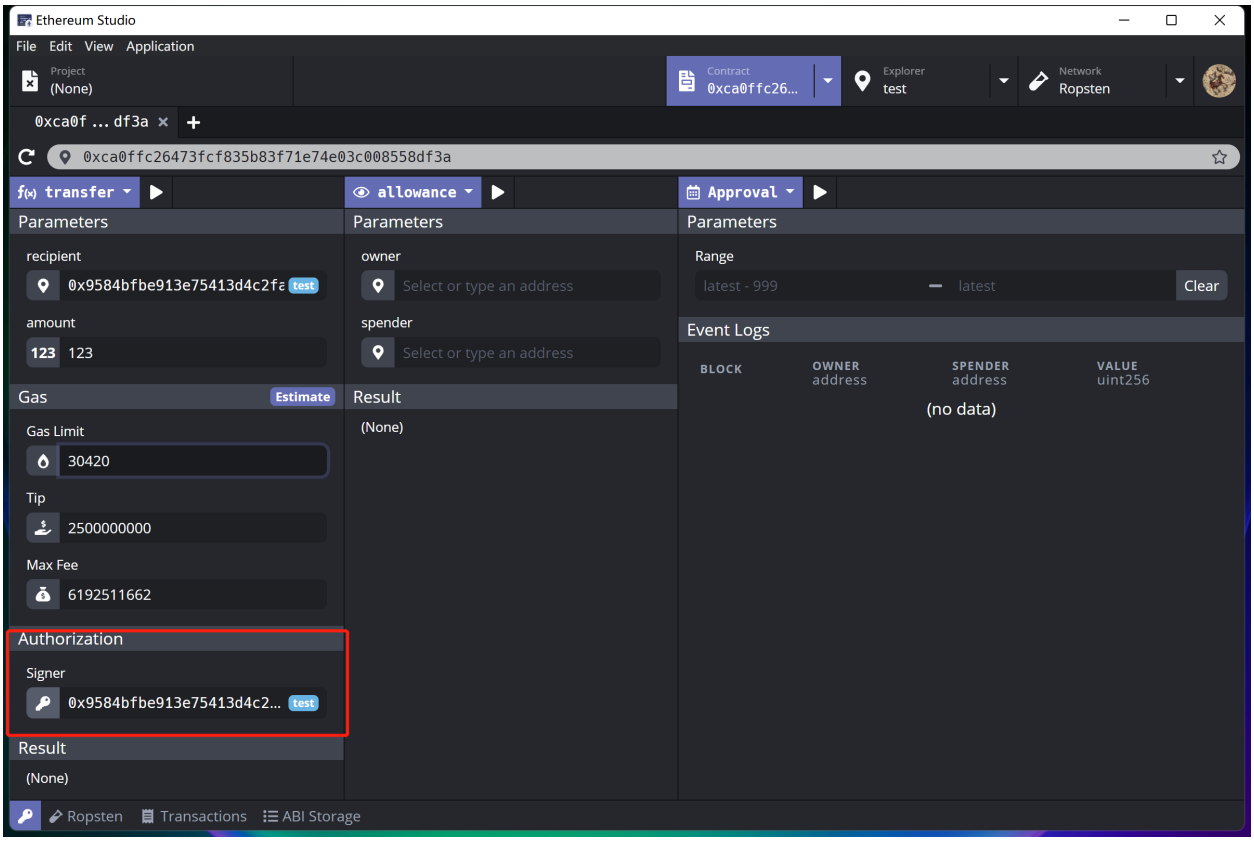
8.1.2 Gas

If developers want to call functions, there would be some gas fee and tip required by the miner of blockchain nodes. Especially when the network is congested, the gas fee can be very large, so developers should set a “Max Fee” in case of an unexpectedly high cost. Developers can click the purple “Estimate” button to get the real-time gas price of deploying contract on network. If developers supposes the price is not fair, they can click again or wait for a period and the Ethereum Studio will show changed price.



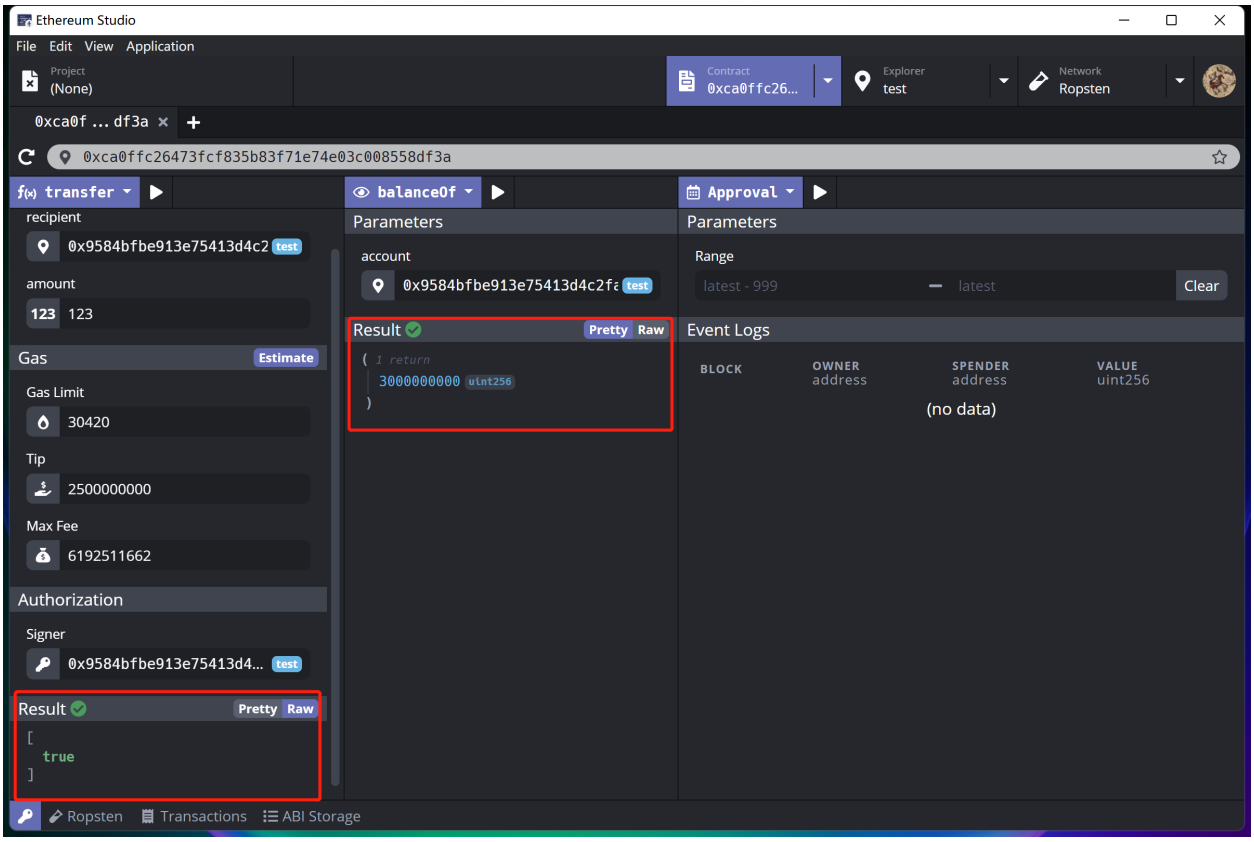
8.1.3 Authorization

In “Authorization”, developers can choose “Signer” to pay gas fee and tip.



8.1.4 Results

In “Results”, there are some direct results of the return value of functions. Developers check error or successful messages through results.



8.2 View Functions

The middle pannel represents “View Functions” in deployed contracts. View functions ensure that they will not modify the state. Most of time, developers can quickly check variable state by calling view functions. Yet view functions still could receive variables and return newly created data structure and variables. So this pannel could help developers check if they get wanted middle results.

8.2.1 Parameters

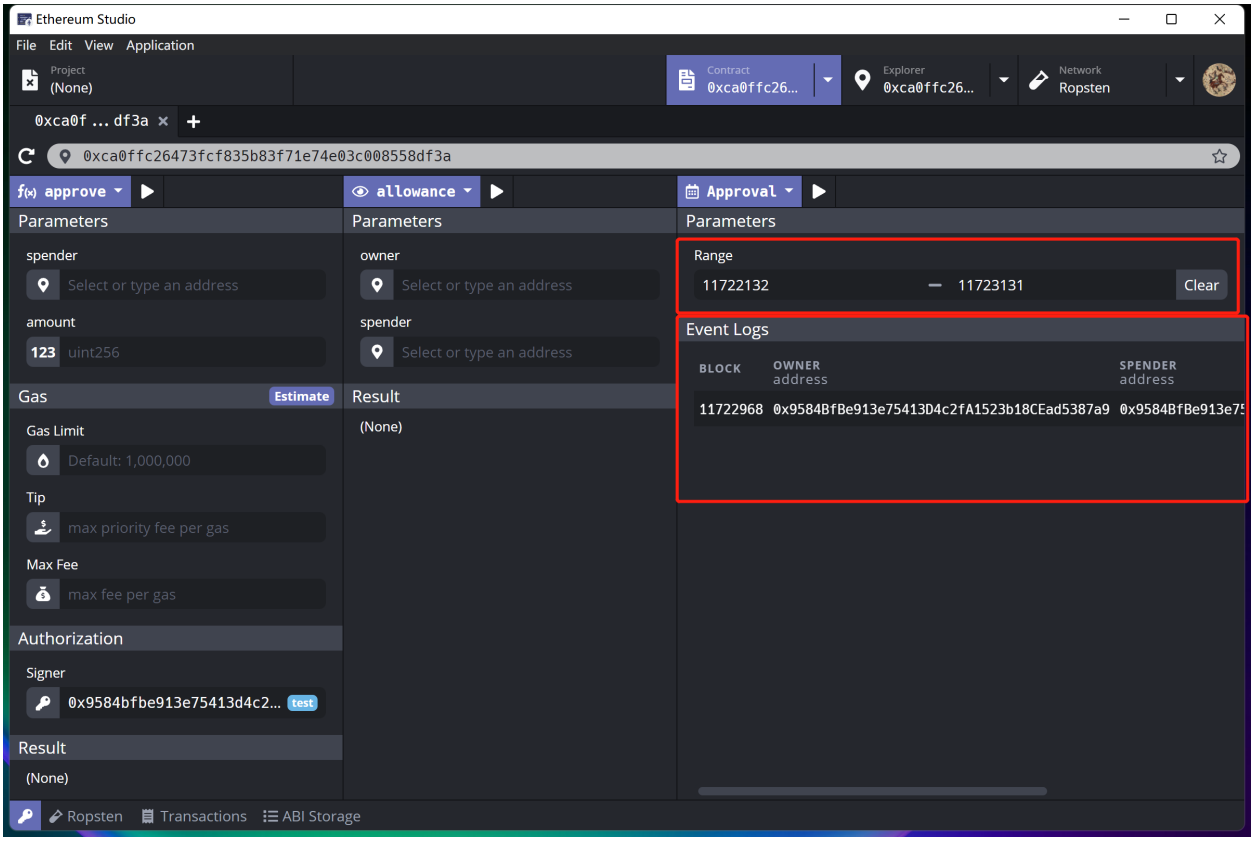
Some of view functions receive variables so they have “Parameters”. In “Parameters”, developers could switch variable types between “hex” bytes or “utf8” uint as functions required. By excuting function, developers check and get desire results without changing variables in contracts.

8.2.2 Results

There are two types of results, “Pretty” and “Raw”. In “Raw”, there are original return values of excuted view function. Switching to “Pretty”, the return values has more detail information such as number and types of returnd value.

8.3 Events

"Events" locates on the right pannel, where developers can choose and check detail information of events on the network. The types of events are corresponding to events in contract. For example, in the deployed ERC20 contract, there are "Approval" and "Transfer" events. Click the "Execute" triangle beside the "Approval" button, developers can check the latest event with its block number in the settled range.



8.3.1 Parameters

In "Parameter", developers set numbers to quickly check the range of events in a list. If developers do not set numbers, the data range is the latest 100 by default.

8.3.2 Events Log

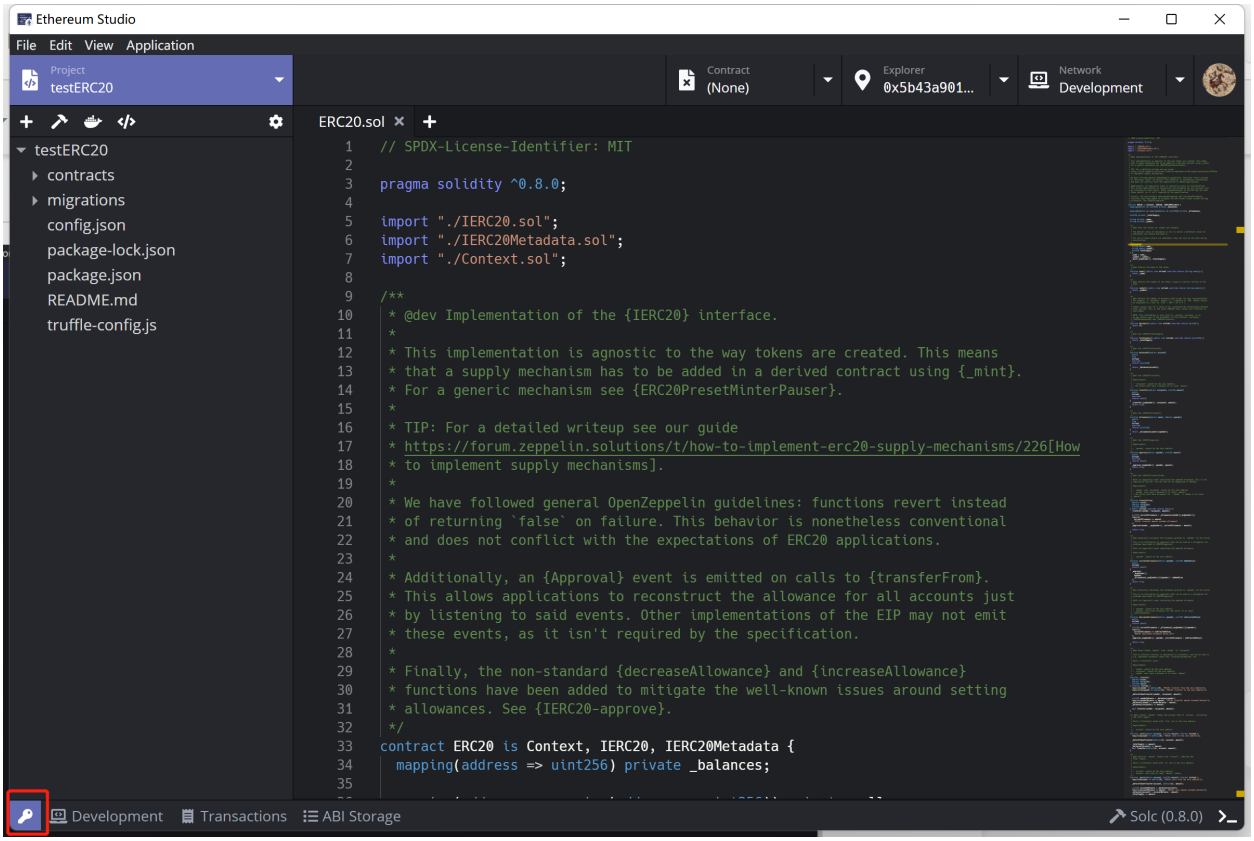
Detail information of required events will show on the "Events Log" with settled parameters and excuted events.

CHAPTER

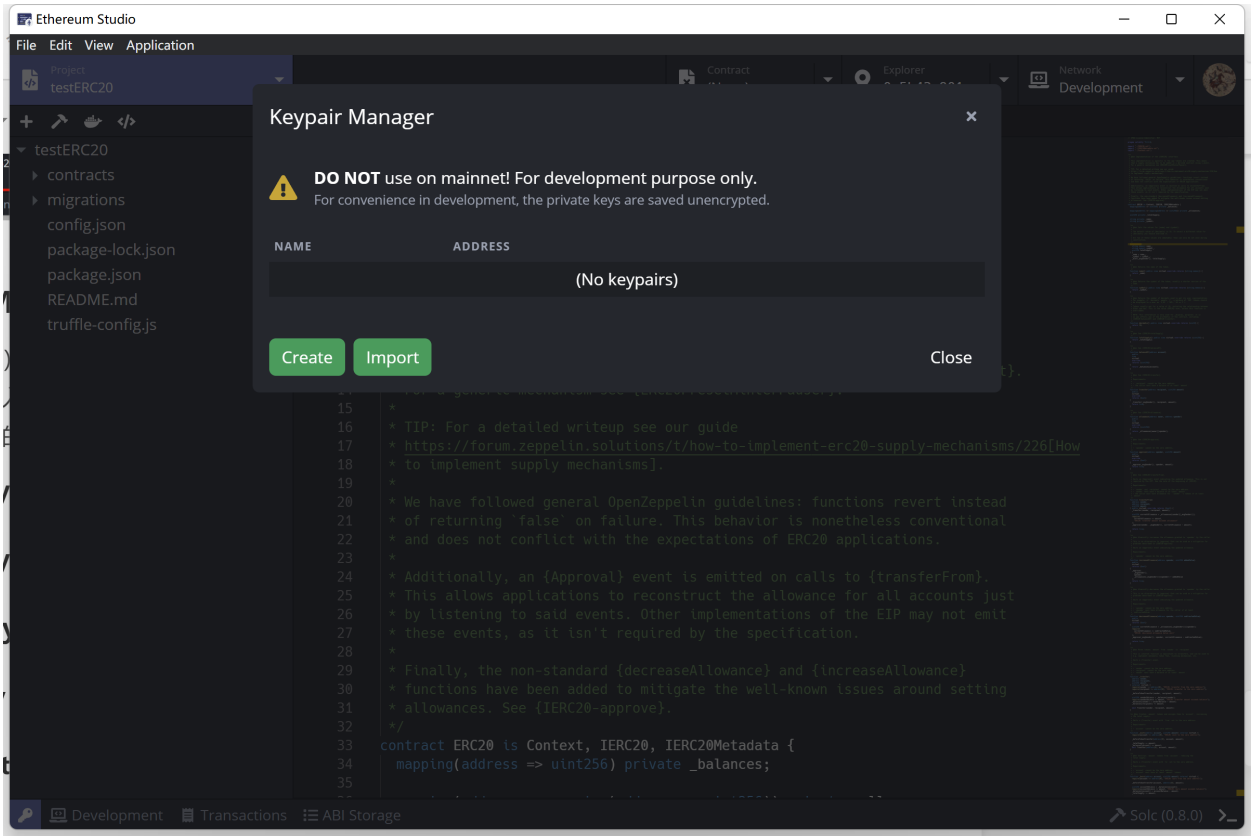
NINE

KEYPAIR MANAGER

Click the purple “key” icon in the bottom left of panel and developers can see the “Keypair Manager” window popup.

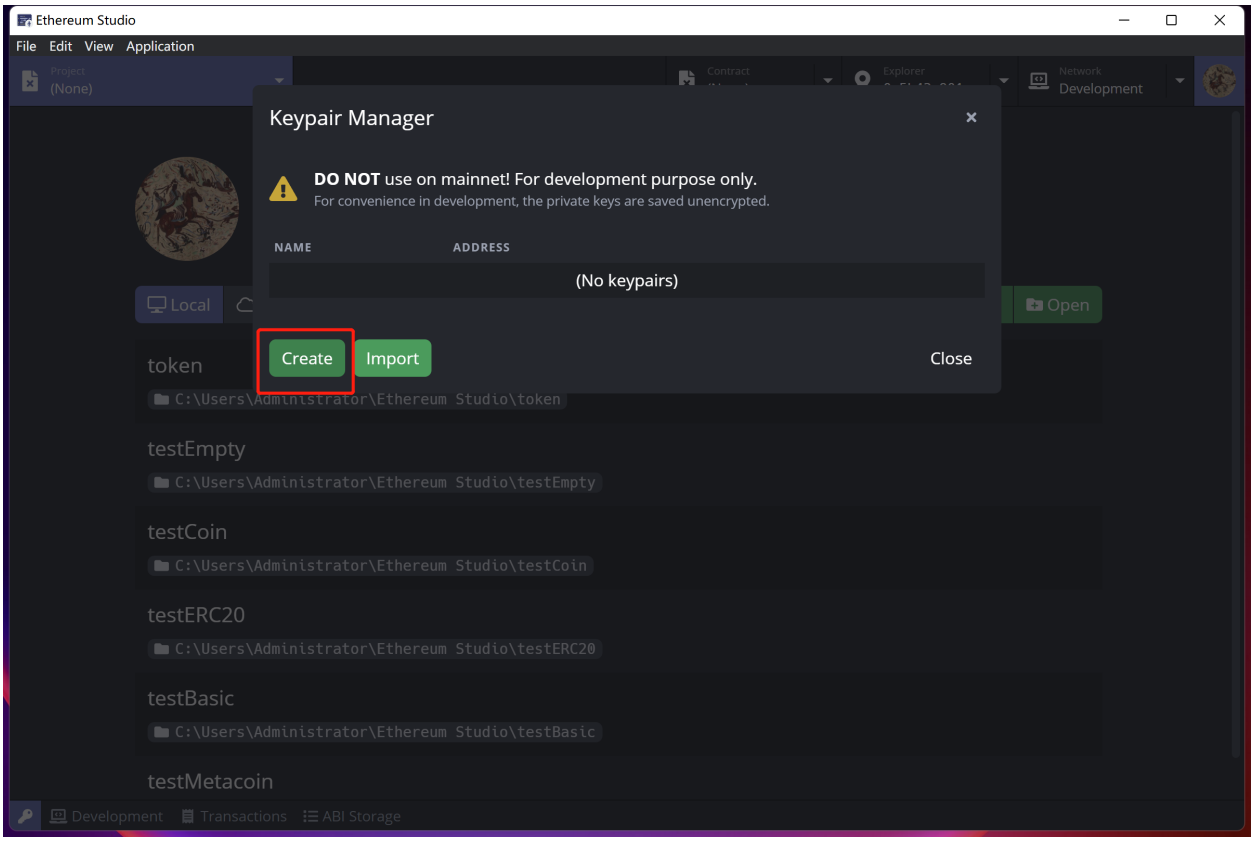


In “Keypair Manager” panel, there is a reminder showing that keypairs kept in the manager should not be used on mainnet. For the convenience of development, the keypairs are all unencrypted, private keys will easily be checked by anyone using a desktop client under the account. Since keypairs used on mainnet containing real ETH assets and private keys, it is dangerous to lose ETH and assets if developers keep Ethereum mainnet keypairs in the keypair manager.



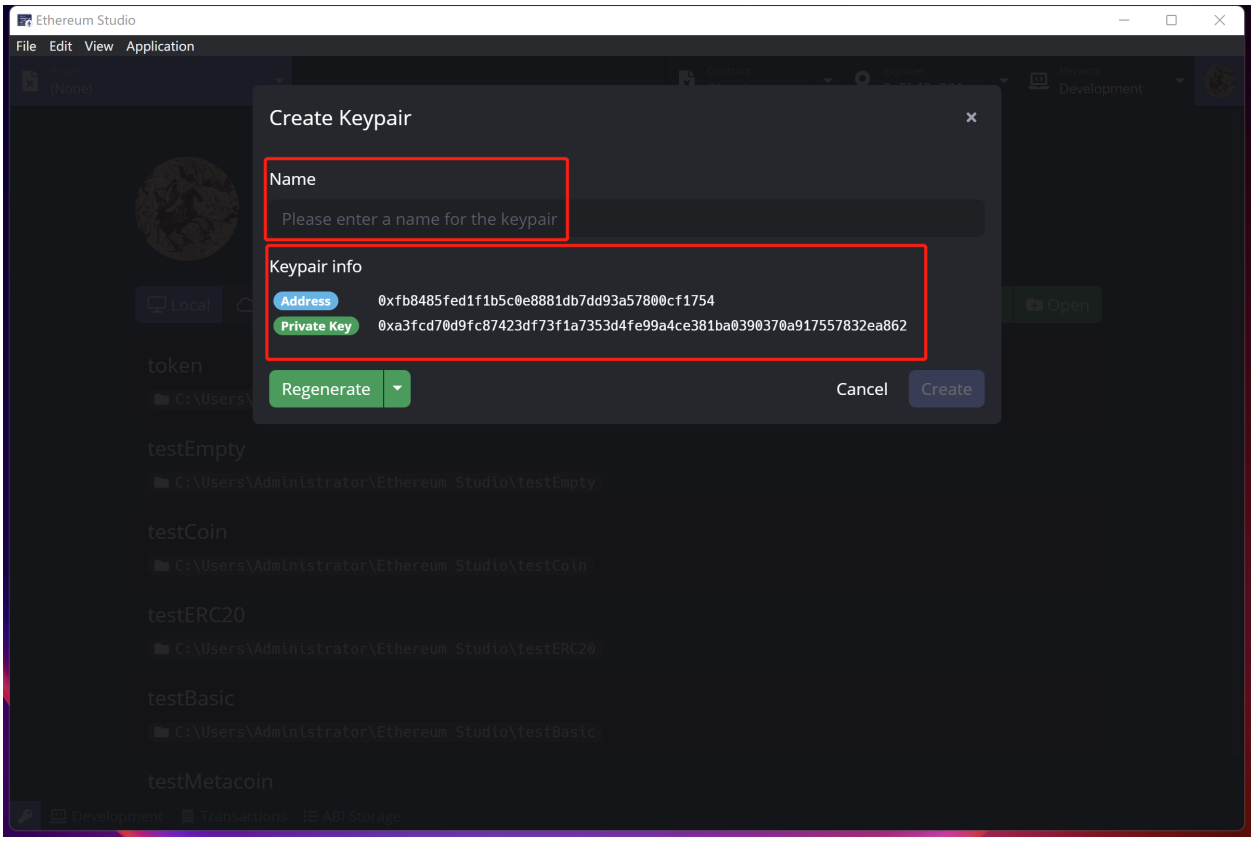
9.1 Create Keypair

Click the "Create" button on the bottom left corner. Developers can easily generate a random keypair for later deployment.



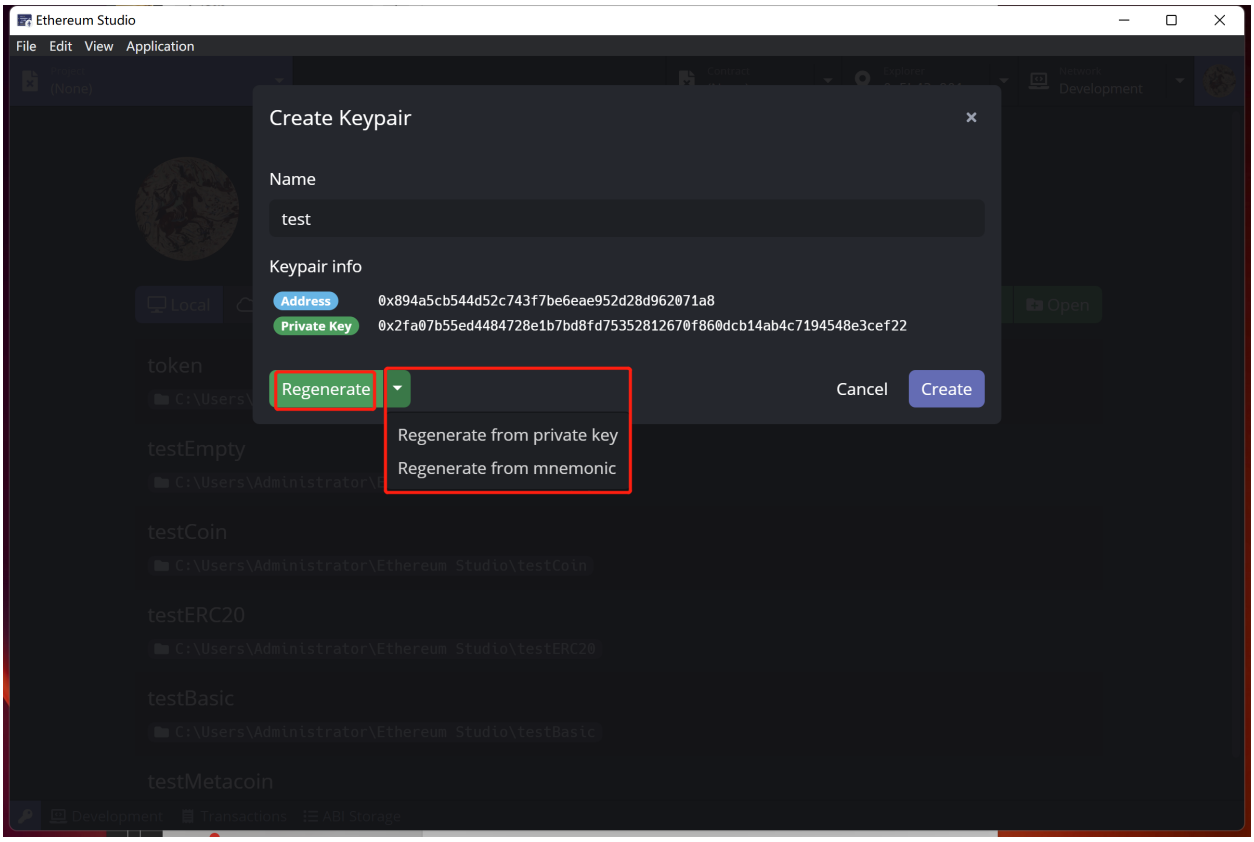
In “Create Keypair” window, there will be a newly generated address and private key in “Keypair info”. The newly created key pair has no name, and developers can input a desired name for it as long as the new name is not the same as any other keypairs name. Otherwise, there would be an error message reminding the name has been used. Names can be easily changed in “Keypair Manager” later.

The keypair will show its name rather than address string in the Block Explorer panel since names may contain a more meaningful message for developers to recognize.



In "Regenerate", an inverted triangle shows that developers can either regenerate a private key or regenerate a mnemonic. In the blockchain, mnemonic means a fixed-length number of words that mnemonic can generate to one and only one private key for its user. Compared with private keys including random permutations of characters and numbers, the mnemonic can be easily remembered by a user who wants to keep critical information in a meaningful way.

If developers wants another keypair, they can click "Regenerate" and have a different private key or mnemonic with a corresponding address. After generating keypair and name, developers click "Create" in the bottom right corner of the panel and save this keypair in "Keypair Manager".

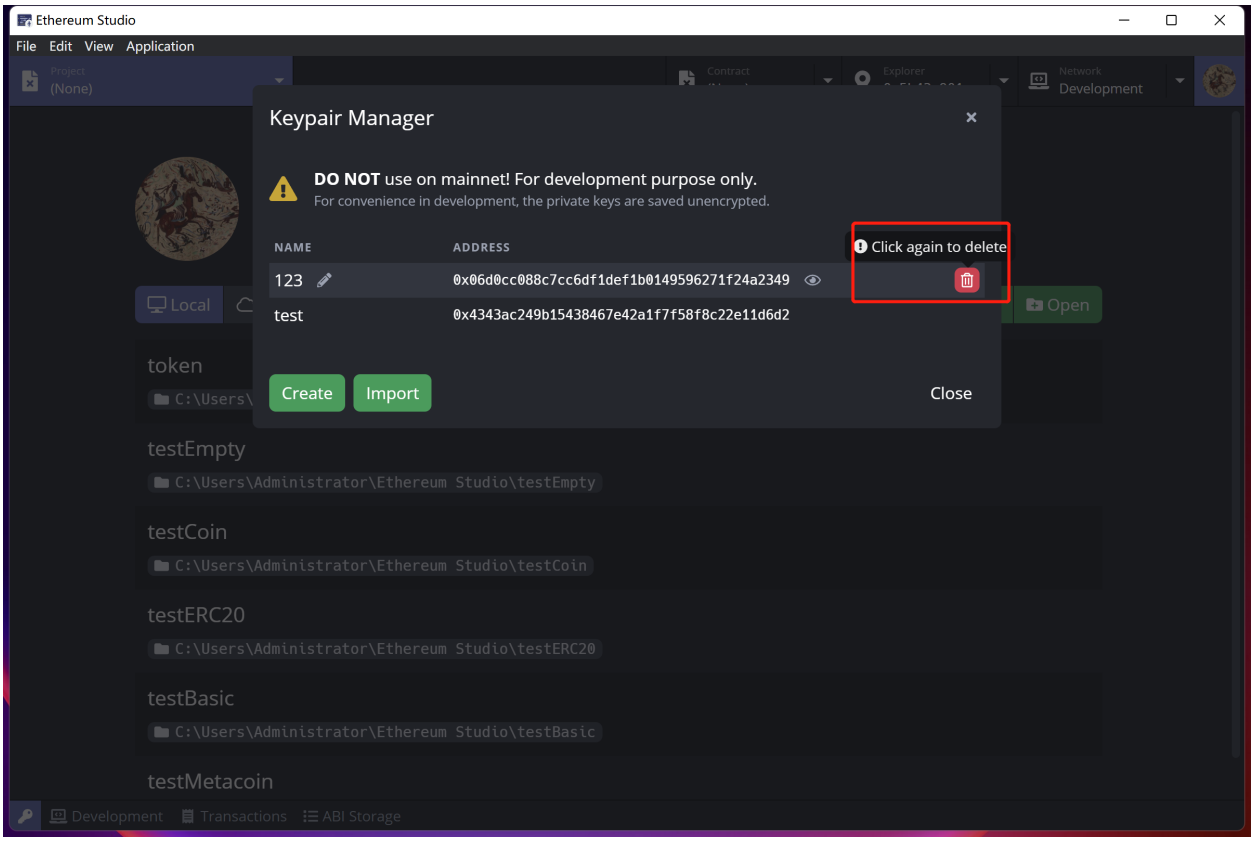


9.2 Check Keypair

In “Keypair Manager”, there is a list of keypairs showing names and adresses. Developers can quickly select whole address by double clicking the address. If developers want to check private key of any keypair, they can move mouse upon the address and there will be a “eye” icon showing on the right of address. Double click the icon and there will be a window named “View Private Key” showing “Address” and “Private Key” of selected keypair.

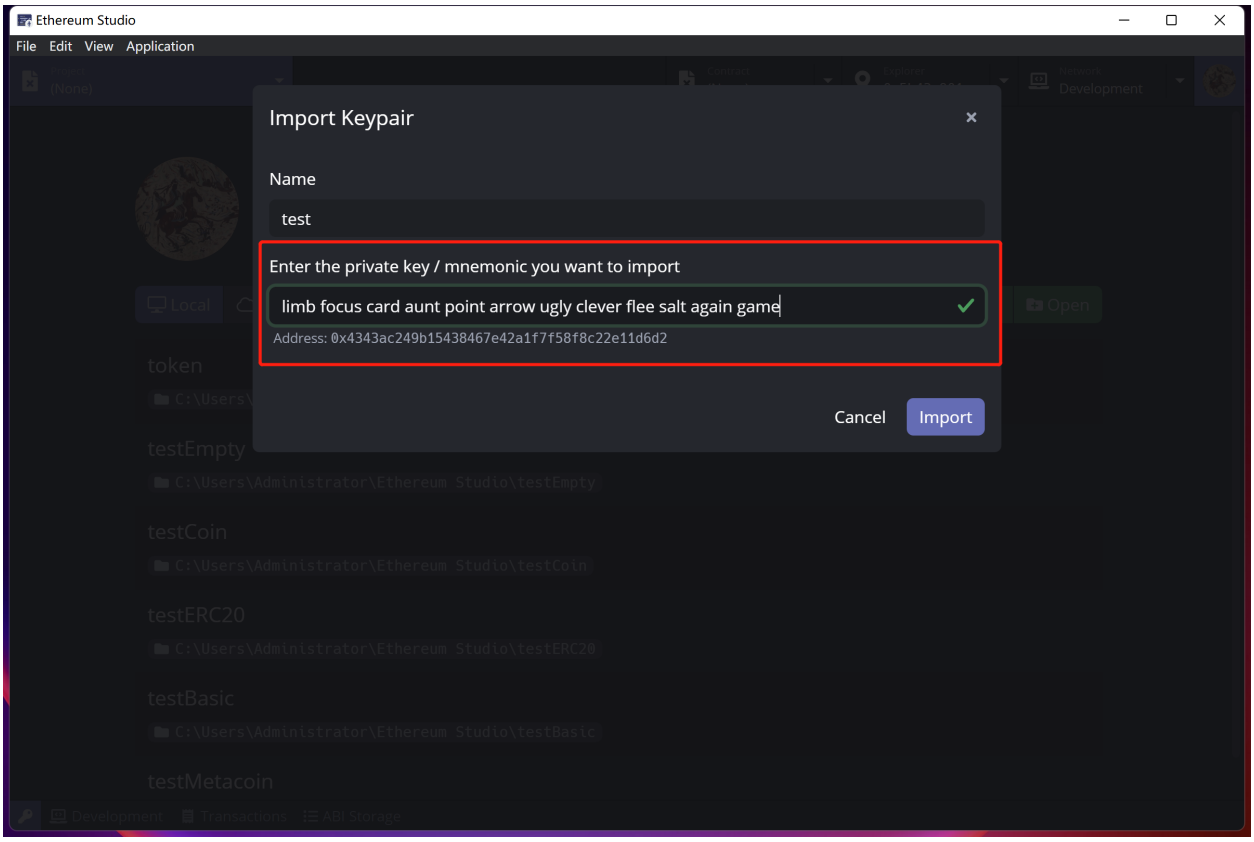
9.3 Delete Keypair

After creating several keypairs, developers can delete the unwanted keypair by double-clicking the “trash can” icon at the right end of the address of keypair. Please remember that the keypair deleted can not be restored unless one has kept the keypair information before and import the keypair again mannually.



9.4 Import Keypair

Except for creating or recreating a random key pair, developers can import an existed keypair by clicking the green "Import" button on the bottom left. There will be an "Import Keypair" window after clicking. Developers can copy the existed keypair information and paste it into the box. The corresponding address is automatically generated after inputting the private key or mnemonic. A green "checkmark" will be at the end of the box, indicating a correct input under rules. Then, click the "Import" button on the bottom right corner to import the keypair.



9.4.1 Import MetaMask Mnemonic

In MetaMask, developers uses a mnemonic as a private key to enter the wallet. Developers can import the mnemonic of MetaMask as a login MetaMask wallet from a newly installed browser extension of MetaMask. Developers can check detail information in link.

CHAPTER

TEN

RPC CLIENT

CHAPTER

ELEVEN

SUPPORTED FAUCETS

There are currencies on the Ethereum network such as ETH and test ETH. Unlike ETH in the mainnet, test ETH has no real value. Therefore, there are no markets for testnet ETH. Most people get testnet ETH from faucets. Faucets are web apps where developers can input an address and the requested test ETH will be sent automatically.

First, choose one of the Ethereum testnets. Then set the account of Block Explorer. There is a "faucet" icon at the right end of the address search bar. Click the icon and it will turn to the faucet page of the corresponding testnet. The Ethereum Studio only supports faucets of Ropsten, Rinkeby and Kovan. Developers can check the Goerli faucet link by oneself.

11.1 Ropsten Faucet

Click the "Faucet" icon and the page will jump to the Ropsten Faucet. Developers can copy the wallet address and paste it into the box. Then developers clicks the button "Send me test Ether" and wait for a few minutes. There will be several test ETH of Ropensten testnet on balance.

- Ropsten Faucet 1
- Ropsten Faucet 2
- Ropsten Faucet 3

11.2 Rinkeby Faucet

Following faucet instruction via inputting specific Twitter or Facebook message links, developers can get test ETH on Rinkeby. Developers can change the amount of test ETH with different lengths of time.

- Rinkeby Faucet 1
- Rinkeby Faucet 2

11.3 Goerli Faucet

Following faucet instruction via inputting specific Twitter or Facebook message links, developers can get test ETH on Rinkeby. Developers can change the amount of test ETH with different lengths of time. This procedure is similar to Rinkeby Faucet.

- Goerli Faucet 1
- Goerli Faucet 2

11.4 Kovan Faucet

Click the link of “Kovan Faucet 1” and request test ETH as instruction.

- Kovan Faucet 1
- Kovan Faucet 2

CHAPTER

TWELVE

SUPPORTED FRAMEWORKS

12.1 Open Zeppelin

Open Zeppelin provides a complete suite of security products to build, manage, and inspect all aspects of software development and operations for Ethereum projects.

In the Ethereum Studio, developers can import Open Zeppelin contracts through the template. Detail information about Open Zeppelin can be checked in link.

12.2 Hardhat

Hardhat is a development environment to compile, deploy, test, and debug Ethereum software. It helps developers manage and automate the recurring tasks inherent to the process of building smart contracts and dApps and quickly introduces more functionality around this workflow. This framework means compiling, running and testing smart contracts at the very core.

In the Ethereum Studio, developers can use Hardhat as a development framework. Developers can choose it during the creation of projects. Developers can also use this framework during the development process by the command line in the Ethereum Studio. Detail information about Hardhat can be checked in link.

12.3 Truffle

Truffle is a development environment, testing framework and asset pipeline for blockchains using the Ethereum Virtual Machine (EVM). With Truffle, users get a built-in smart contract compilation, linking, deployment and binary management. Truffle also has network management for deploying to any number of public & private networks.

In the Ethereum Studio, developers can use Hardhat as a development framework. Developers can choose it during the creation of projects. Developers can also use this framework during the development process by the command line in the Ethereum Studio. Detailed information about Hardhat can be checked in link.

Dockerized Truffle used only Truffle in Docker image without other dependencies like Node.js and Npm in local. The other two Frameworks, Hardhat and Waffle, require Node.js and Npm.

12.4 Waffle

Waffle is a library for writing and testing smart contracts. Sweeter, simpler and faster than Truffle. In Waffle, “Simpler” means minimalistic, few dependencies; “Sweeter” means nice syntax, easy to extend; “Faster” means to focus on the speed of tests execution. Waffle uses a set of chai matches, and it can import contracts from npm modules easily. There is also a fast compilation with native and dockerized solidity contracts. It provides fixtures that help write fast and maintainable test suites.

In the Ethereum Studio, developers can use Waffle as a development framework. Developers choose Waffle during the creation of projects. Developers can also use this framework during the development process by the command line in the Ethereum Studio. Detailed information about Waffle can be checked in link.

CHAPTER

THIRTEEN

SUPPORTED TESTNETS

Ethereum Mainnet is the primary public Ethereum production blockchain, where actual-value transactions occur on the distributed ledger. There are many different Ethereum Testnets, and each testnet uses its own test ETH as currency, respectively. It's generally essential to test all smart contracts code on a testnet before deploying it to the Mainnet.

Most testnets use a proof-of-authority consensus mechanism. This means a small number of nodes are chosen to validate transactions and create new blocks – staking their identity in the process. It's hard to incentivize mining on a proof-of-work testnet which can leave it vulnerable.

13.1 Ropsten

Ropsten is a proof-of-work testnet for those running Geth, Besu and all other Ethereum clients. This means it's the best like-for-like representation of Ethereum. Ropsten started in November 2016 and it can be used with all clients. Detail information can be checked in link.

13.2 Rinkeby

Rinkeby is a proof-of-authority(clique) testnet for those running Geth, Besu, Nethermind, and OpenEthereum client. Rinkeby started in April 2017 and is immune to spam attacks(as Trusted parties control ether supply). Detail information can be checked in link.

13.3 Goerli

Goerli is a proof-of-authority testnet that works across clients. Goerli started in November 2018. Goerli doesn't fully reproduce the current production environment as it uses PoA. Detail information can be checked in link.

13.4 Kovan

Kovan is a proof-of-authority testnet for those running OpenEthereum clients. Kovan started in March 2017 and is immune to spam attacks. Kovan doesn't fully reproduce the current production environment as it uses PoA. Detail information can be checked in link.

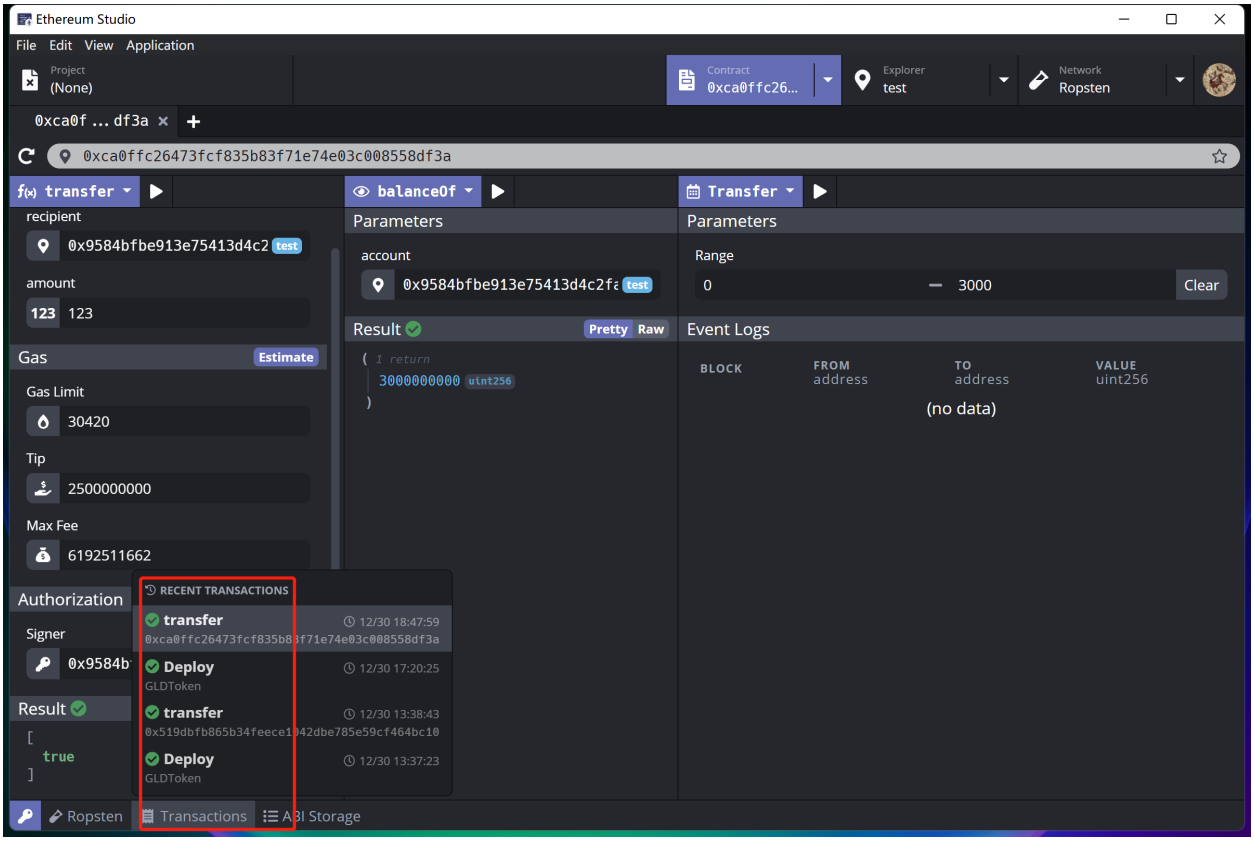
CHAPTER

FOURTEEN

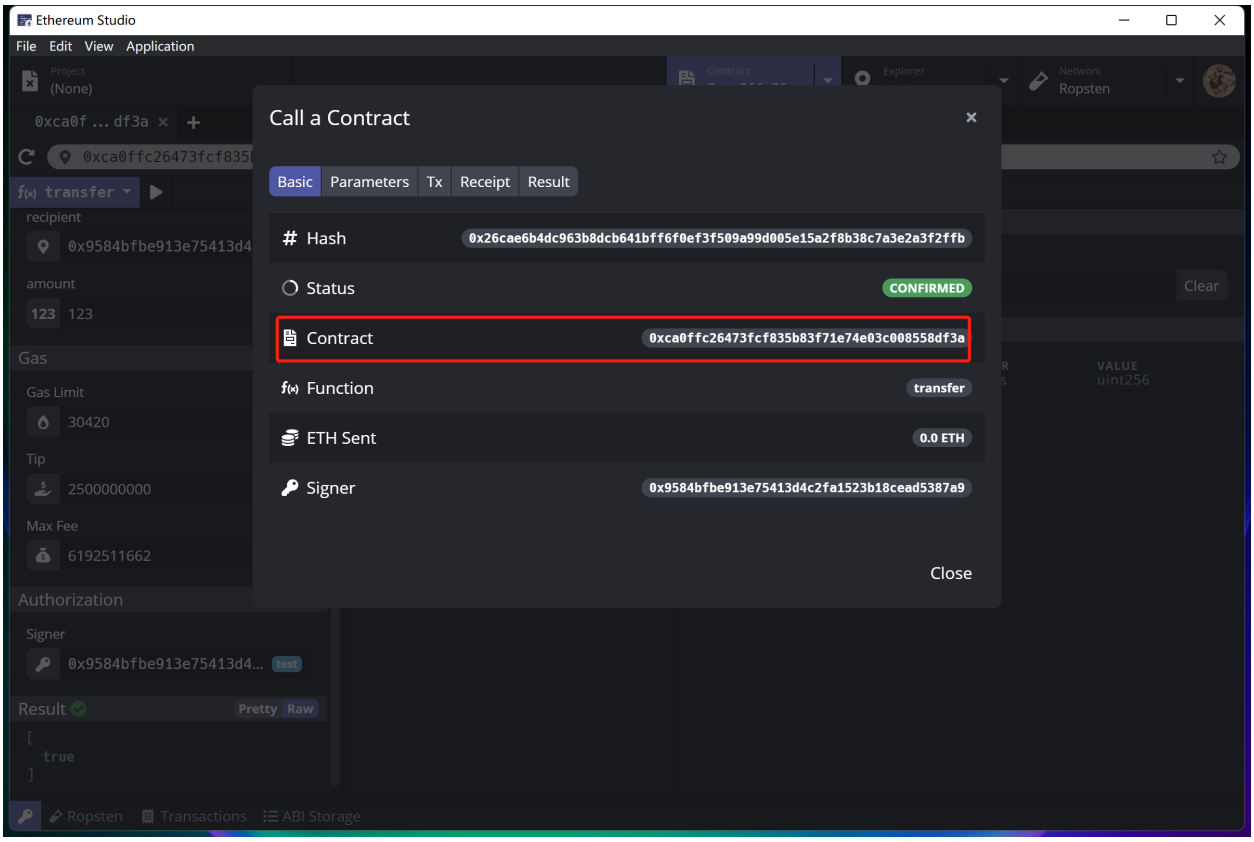
TRANSACTION HISTORY

14.1 Transaction Detail

After calling functions, developers can check the transaction status in the "Transaction" button at the bottom of IDE. Click the function name, and there will be a popup "Call a Contract" window.



There is detailed information about the transaction in the "Call a Contract" window. In this window, the contract address means where the function is called. There is other detailed information for developers to check in different panels.


Ethereum Studio
File Edit View Application
Call a Contract
Basic Parameters Tx Receipt Result
Hash 0x26cae6b4dc963b8dcb641bff6f0ef3f509a99d005e15a2f8b38c7a3e2a3f2ffb
Status CONFIRMED
Contract 0xca0ffc26473fcf835b83f71e74e03c008558df3a
Function transfer
ETH Sent 0.0 ETH
Signer 0x9584bfbe913e75413d4c2fa1523b18cead5387a9
Close

CHAPTER

FIFTEEN

TRUFFLE MIGRATION SCRIPT

CHAPTER

SIXTEEN

ABI STORAGE