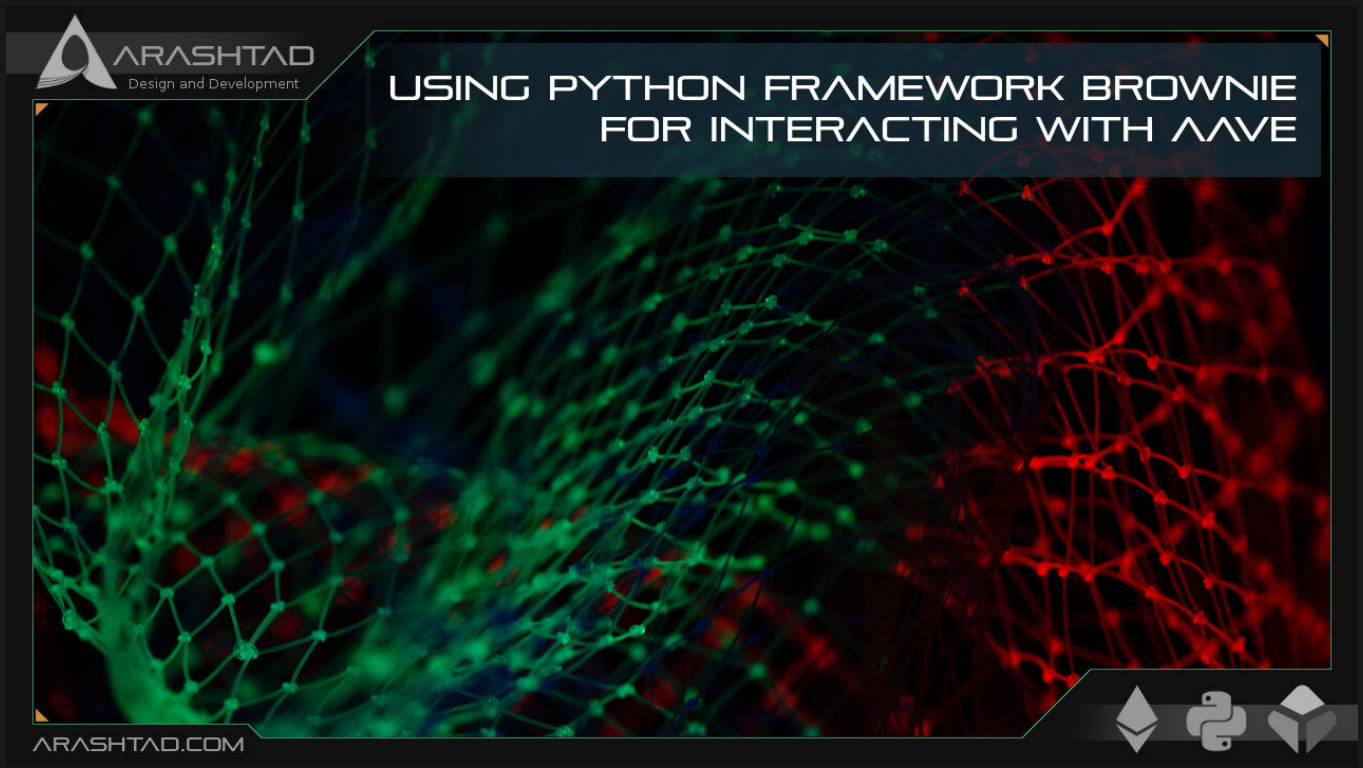
Using Python Framework Brownie for Interacting with Aave Protocol

No comments



In this tutorial, regarding the Aave protocol, we are going to interact with the Aave protocol using Brownie. The first step is to swap ETH with WETH which is necessary when we want to deal with Aave. This lending and borrowing protocol, accept WETH instead of ETH. We add WETH (Wrapped Ethereum) to our Metamask wallet and also create the project folder for interacting with the Aave protocol. After the first (which was swapping ETH with WETH), it is now time to deposit our WETH. To do this, we will download some solidity scripts related to the Aave lending pool, and paste them into the contracts directory of the project folder. Finally, modify the deploy.py to also activate the depositing contracts.

Interacting with Aave Protocol by Using Brownie

This series of tutorials is the continuation of How to deploy a smart contract using python web3 tools in addition to solidity, smart contracts, and brownie python tools. As a result, it is highly recommended that before you start this tutorial, be familiar with the python web3 and smart contracts. In the last tutorial, we learned how to interact with the

Aave protocol using the interface provided at this link. In this part, we are going to do the same interactions using python scripts.
These interactions include:

1. Swapping ETH with WETH (Wrapped ETH).

2. Depositing WETH into Aave.

3 Using WETH collateral to borrow DAI

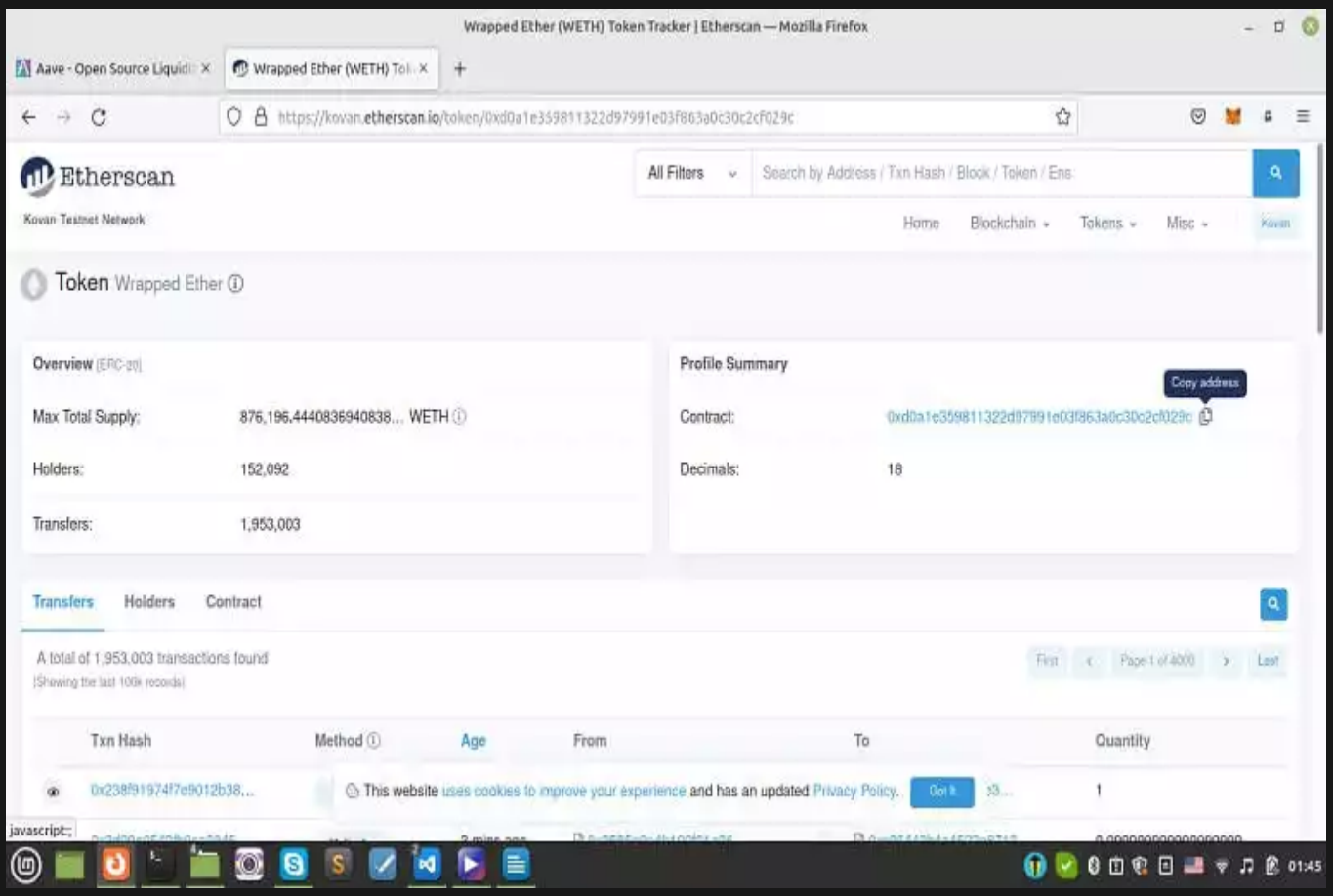
4. Paying back the borrowed DAI

Creating the Project

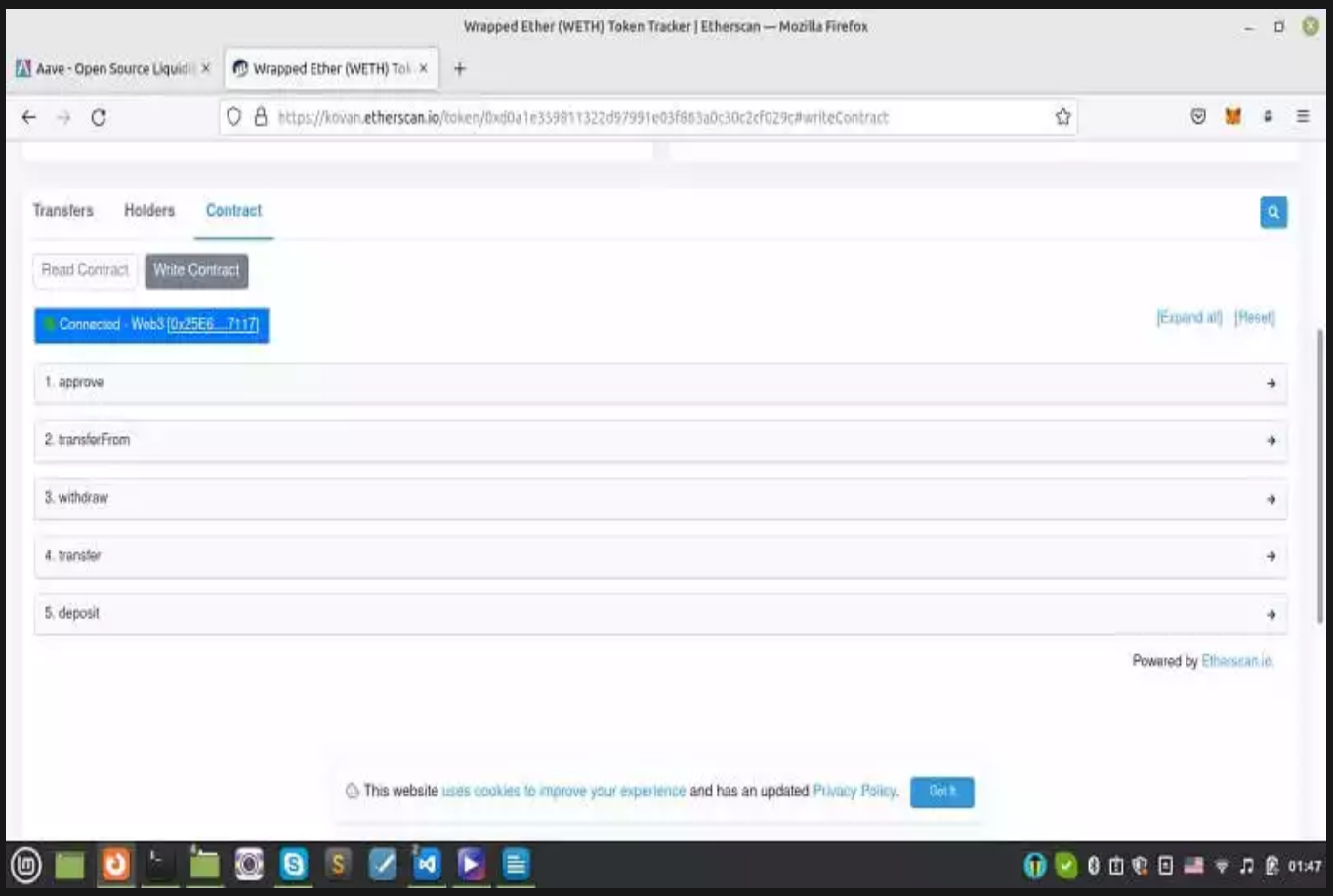
So, like every project, we begin with:

```
mkdir aave_brownie
cd aave_brownie
code .
Brownie init
```

The first thing you should do to interact with the Aave protocol is to deposit some ETH into Aave. When you do this, the actual thing that happens is that ETH is swapped into the ERC-20 version of Ethereum, called WETH (Wrapped Ethereum). Because in our project we use the Kovan network, we should copy the address of the contract from this link and paste it into our config file (brownie-config.yaml).



If you go to write the contract, you will see an actual interface of the WETH contract for deposit and withdrawal.



Now, let's write our scripts, we begin with creating a file called get_weth.py in the scripts folder:

```
from brownie import accounts, config, network, interface
from scripts.helpful_scripts import get_account

def main():
 get_weth()

def get_weth(account=None):
 account = get_account()
 weth =interface.Weth(
 config["networks"][network.show_active()]["weth_token"])
 tx = weth.deposit({"from": account, "value": 0.1 * 1e18})
print("Received 0.1 WETH")
return tx
```

Use the helpful_scripts.py to get the proper account in the scripts folder:

```
from brownie import network, config, accounts
def get_account():
if network.show_active() in ["hardhat", "development", "mainnet-fork"
]:
return accounts[0]
if network.show_active() in config["networks"]:
  account = accounts.add(config["wallets"]["from_key"])
return account
return None
```

```
pragma solidity ^ 0.4.19;

interface Weth {
function allowance(address owner, address spender) external view
returns (uint256 remaining);
function approve(address spender, uint256 value) external returns (
bool success);
function balanceOf(address owner) external view returns
 (uint256 balance);
function decimals() external view returns (uint8 decimalPlaces);
function name() external view returns (string memory tokenName);
function symbol() external view returns (string memory tokenSymbol);
function totalSupply() external view returns
 (uint256 totalTokensIssued);
function transfer(address to, uint256 value) external returns (bool
 success);
function transferFrom(address from, address to, uint256 value)
external returns(bool success);
function deposit() external;
function withdraw(uint wad) external;
}
```

And an interface named IWeth.sol in the interface folder (you can notice that it has the methods and events of an ERC20 token):

```
dotenv: .env
networks:
 mainnet-fork:
  weth_token:"0xc02aaa39b223fe8d0a0e5c4f27ead9083c756cc2"

 kovan:
  weth_token:"0xd0a1e359811322d97991e03f863a0c30c2cf029c"
wallets:
 from_key:${PRIVATE_KEY}
```

Also, enter your private key and Infura ID in the .env file, in the main directory:

```
export WEB3_INFURA_PROJECT_ID=' your Infura ID'
export PRIVATE_KEY=' Your private key '
```

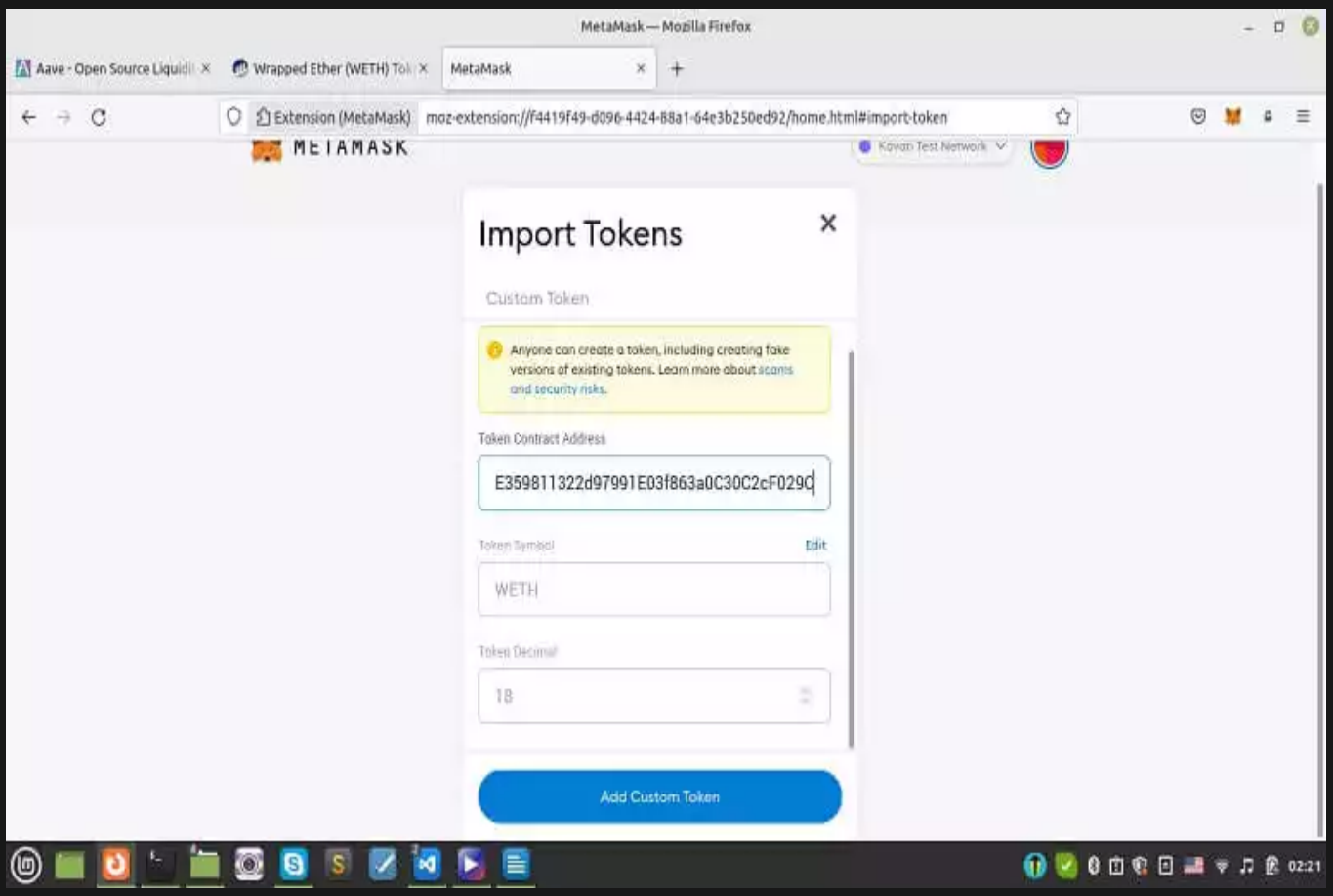
It is now time to run our project on the Kovan test network to swap ETH with WETH:

```
brownie run scripts/get_weth.py --network kovan
```

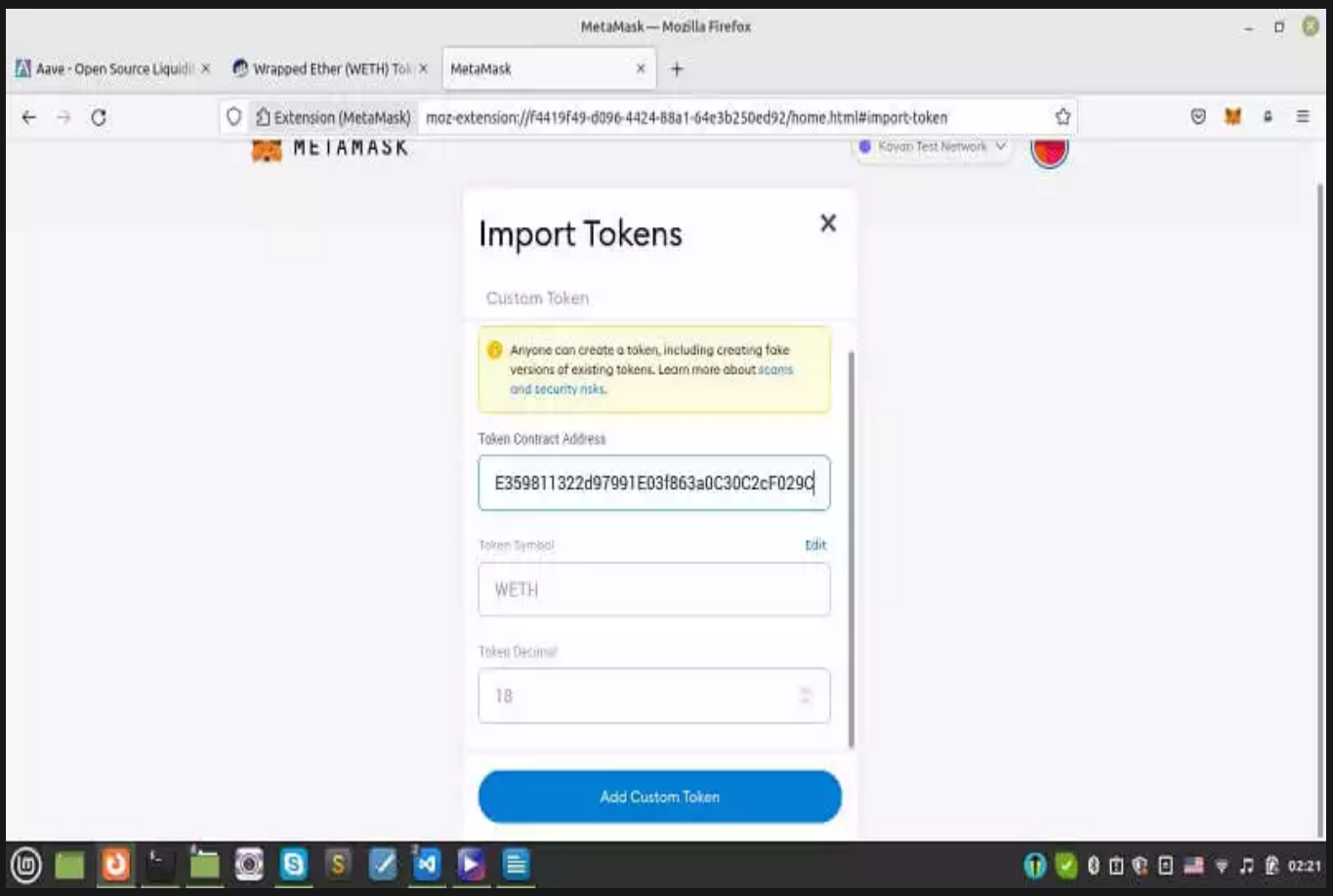
Result:

```
Brownie v1.18.1 - Python development framework for
EthereumAaveBrownieProject is the active project.Running
'scripts/get_weth.py::main'... Transaction sent:
0x96955d6833e1ceb11e8de71d8c64cffd4af84765ceb7c333e6fa666c2f2e21f5 Gas
price: 2.500000007 gwei Gas limit: 49572 Nonce: 7 WETH9_.deposit
confirmed Block: 30924833 Gas used: 45066 (90.91%)Received 0.1 WETH
```

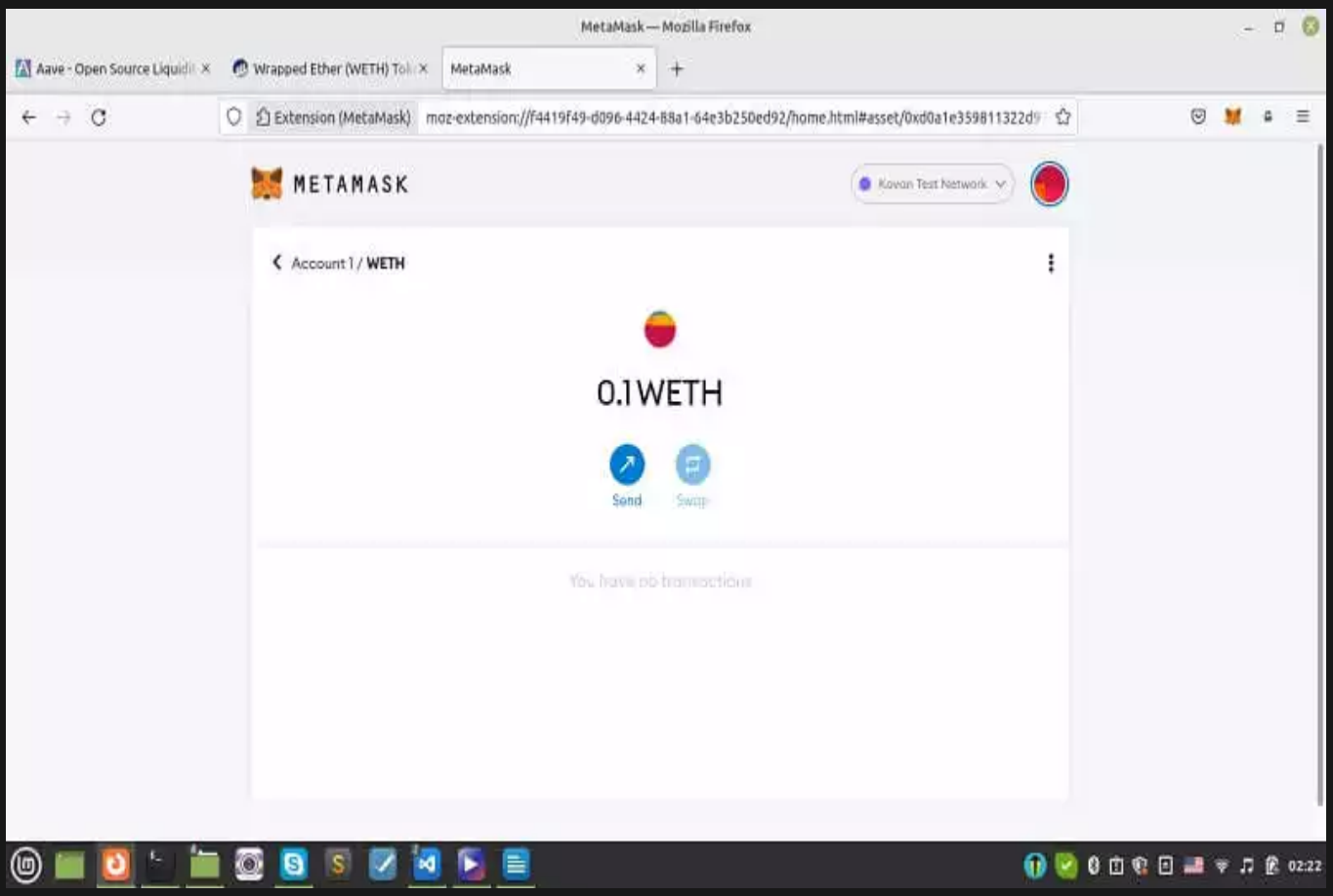
To make sure we have actually received the desired WETH, let's head over to our Metamask and import WETH token by pasting the address of WETH Kovan contract in the Token Contract Address box:



And there we go! We have 0.1 WETH in our account in exchange for 0.1 ETH.



And there we go! We have 0.1 WETH in our account in exchange for 0.1 ETH.



After our first approach towards using Brownie for Aave protocol (which was swapping ETH with WETH), it is now time to deposit our WETH. To do this, we will download some solidity scripts related to the Aave lending pool, and paste them into the contacts directory of the project folder. And finally, modify the deploy.py to also activate the depositing contracts.

Depositing WETH

After having successfully swapped ETH with WETH, it is time to deposit our WETH to the lending pool, so let's keep the scripts we added to our project, and add some more files to be able to do this task. First and foremost, we need to export our Etherscan API key in the .env file:

```
export ETHERSCAN='Your Etherscan API key'
```

The next thing we need to do is to add some interfaces to the interfaces folder from the Aave GitHub repository. These files include:

1. ILendingPoolAddressesProvider.sol

2. ILendingPool.sol

Notice that you should modify lines 5 and 6 in the code to be able to use it in your directory:

```
import {ILendingPoolAddressesProvider} from
'@aave/contracts/interfaces/IlendingPoolAddressesProvider.sol';
import {DataTypes} from
'@aave/contracts/protocol/libraries/types/DataTypes.sol';
```

Besides, in the brownie-config.yaml, add these lines:

```
dependencies:
 - smartcontractkit/chainlink-brownie-contracts@04.0
 - aave/protocol-v2@10.1
compiler:
 solc:
  remappings:
  "@chainlink=smartcontractkit/chainlink-brownie-contracts@0.4.0"
  "@aave=aave/protocol-v2@1.0.1"
```

IERC20.sol

We also need to paste in ERC-20 smart contract in the contracts directory:

```
pragma solidity ^ 0.6.6;

interface IERC20 {
function allowance(address owner, address spender) external
 view returns (uint256 remaining);
function approve(address spender, uint256 value) external returns (
bool success);
function balanceOf(address owner) external
 view returns (uint256 balance);
function decimals() external view returns (uint8 decimalPlaces);
function decreaseApproval(address spender, uint256 addedValue)
external returns (bool success);
function increaseApproval(address spender, uint256 subtractedValue)
external;
function name() external view returns (string memory tokenName);
function symbol() external view returns (string memory tokenSymbol);
```

```
function totalSupply() external
 view returns (uint256 totalTokensIssued);
function transfer(address to, uint256 value) external returns (bool
 success);
function transferFrom(address from, address to, uint256 value)
external returns (bool success);
}
```

brownie_config.yaml:

In the brownie-config.yaml file, we also need to add some specifications about the networks:

```
networks:
 default:mainnet-fork
 mainnet-fork:
  lending_pool_addresses_provider:
"0xB53C1a33016B2DC2fF3653530bfF1848a515c8c5"
  weth_token":0xc02aaa39b223fe8d0a0e5c4f27ead9083c756cc2"
 kovan:
  lending_pool_addresses_provider:
"0x88757f2f99175387ab4c6a4b3067c77a695b0349"
  weth_token":0xd0a1e359811322d97991e03f863a0c30c2cf029c"
```

aave_borrow.py

And finally we create a file called aave_borrow.py to start our depositing process:

```
from brownie import accounts, config, interface, network
from web3 import Web3
from scripts.get_weth import get_weth
from scripts.helpful_scripts import get_account

amount = Web3.toWei(0.1, "ether")
```

The above function will convert 0.1 ether into a Wei unit. We use this conversion to avoid writing many zeros after 1.

```
def main():
 account =get_account()
 erc20_address = config["networks"][network.show_active()]["weth_token"
]
```

```
 if networkshow_active() in ["mainnet-fork"]:
get_weth(account=account)
 lending_pool =get_lending_pool()
approve_erc20(amount, lending_pool.address, erc20_address, account)
print("Depositing...")
 lending_pooldeposit(erc20_address, amount, account.address, 0, {"from
": account})
 print("Deposited!")
```

The above function will get the account address depending on the active network and then gets the ERC-20 address of the WETH token according to the chosen network and then gets some WETH token if the Mainnet-fork network is active (remember in the last tutorial we got some WETH in the Kovan network but not in the Mainnet-fork). After that, we will get the lending pool using the function get_lending_pool(). We also approve the ERC-20 transaction using the approve_erc20 function and in the end, we deposit the WETH in the lending pool.

```
def get_lending_pool():
 lending_pool_addresses_provider= interface
.ILendingPoolAddressesProvider(
  config["networks"][network.show_active()][
"lending_pool_addresses_provider"])
 lending_pool_address = lend-ing_pool_addresses_provider.getLendingPool()
 lending_pool =interface.ILendingPool(lending_pool_address)
return lending_pool
```

The above function uses the address of the lending pool according to the active network and then returns the lending pool according to the address. Notice that we use the function of the ILendingPoolAddressesProvider.sol file.

```
def approve_erc20(amount, lending_pool_address, erc20_address, account
):
 print("Approving ERC20...")
 erc20 = interfaceIERC20(erc20_address)
 tx_hash = erc20.approve(lending_pool_address, amount, {"from"
: account})
 tx_hash.wait(1)
 print("Approved!")
 return True
```

The above function will approve the ERC-20 using the IERC20.sol file.

```
if __name__ == "__main__":
```

```
main()
```

Compiling and Running the Project

After having set all files up, it is time to compile our project:

```
brownie compile
```

Result:

```
Brownie v1.18.1 - Python development framework for EthereumGenerating
interface ABIs... Project has been compiled. Build artifacts saved at
/home/mohamad/AAVE_brownie/build/contracts
```

Finally, we deposit the WETH into the lending pool on the Kovan network.

```
brownie run scripts/aave_borrow.py --network kovan
```

Result:

```
Brownie v1.18.1 - Python development framework for
EthereumAaveBrownieProject is the active project.Running
'scripts/aave_borrow.py::main'... Approving ERC20... Transaction sent:
0x0ea015becac17bb9f7a40dd6f93fbe8dd2043c8c41f35baf2158e5c9d58ce73a Gas
price: 2.500000007 gwei Gas limit: 28805 Nonce: 12 WETH9_.approve
confirmed Block: 30939584 Gas used: 26187 (90.91%)WETH9_.approve
confirmed Block: 30939584 Gas used: 26187 (90.91%)Approved!
Depositing... Transaction sent:
0x83cbdafc218186ee61a02097cd14dff42cce5dc1df4f2d66518d1b6fc6ab6501Fetching
source of 0x2646FcF7F0AbB1ff279ED9845AdE04019C907EBE from api-
kovan.etherscan.io... Gas price: 2.500000007 gwei Gas limit: 207906
Nonce: 13 Transaction confirmed Block: 30939586 Gas used: 176916
(85.09%)Deposited!
```

As you can see, we have successfully deposited our 0.1 WETH into the Aave lending pool. In the next part, we are going to borrow some DAI from the pool.

Conclusion:

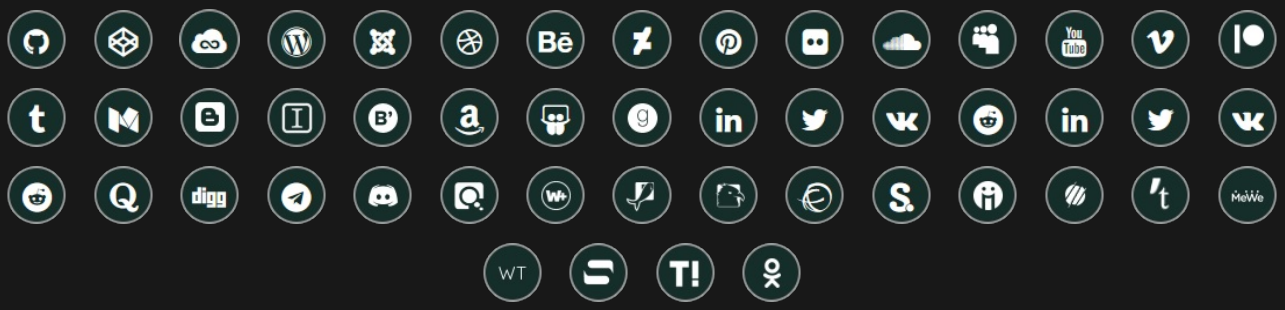
In this tutorial, we have managed to interact with the Aave protocol using Brownie. The first thing we did was to swap ETH with WETH which was a necessary step when we want to deal with Aave. This lending and borrowing protocol accepts WETH instead of ETH. We added WETH (Wrapped Ethereum) to our Metamask wallet and also created the project folder for interacting with the Aave protocol.

In this tutorial, we have downloaded the lending pool smart contract from the Aave protocol GitHub account and pasted them into our directory. In addition to that, we have modified the deploy.py file to deploy the depositing smart contracts as well as adding the aave_borrow.py file, and eventually, we have deposited the WETH we have in the Aave lending pool.

Join Arashtad Community

Follow Arashtad on Social Media

We provide variety of content, products, services, tools, tutorials, etc. Each social profile according to its features and purpose can cover only one or few parts of our updates. We can not upload our videos on SoundCloud or provide our eBooks on Youtube. So, for not missing any high quality original content that we provide on various social networks, make sure you follow us on as many social networks as you're active in. You can find out Arashtad's profiles on different social media services.



Get Even Closer!

Did you know that only one universal Arashtad account makes you able to log into all Arashtad network at once? Creating an Arashtad account is free. Why not to try it? Also, we have regular updates on our newsletter and feed entries. Use all these benefitial free features to get more involved with the community and enjoy the many products, services, tools, tutorials, etc. that we provide frequently.

SIGN UP | NEWSLETTER | RSS FEED