

Using Brownie for Interacting with Aave Protocol (Borrowing & Repaying)

No comments



*This article is about what to do for using **Brownie for Aave protocol** after depositing **WETH** in the Aave lending pool. In our next step, we will try to borrow some **Dai**. To do this, we should add some dependency contracts to the contracts folder such as **AggregatorV3Interface.sol** and **LinkTokenInterface.sol**. We should also modify the **deploy.py** and **brownie-config.yaml** files to adapt to our new functionality.*

Adding the necessary contracts to the contracts folder

In this 3rd part of the tutorial, after depositing ETH into the pool, it is time to borrow some DAI using our WETH collateral.

There are 2 interfaces to add to the interfaces folder:

1. [AggregatorV3Interface.sol](#)
2. [LinkTokenInterface.sol](#)

```
pragma solidity ^ 0.6.6;

interface LinkTokenInterface {
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 transferAndCall(address to, uint256 value, bytes calldata
data) external returns (bool
success);
function transferFrom(address from, address to, uint256 value)
external returns (bool success);
}
```

Modifying the Deploy.py

In the last part we went up to depositing WETH.

```
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!")
```

We continue the def main () like this to borrow DAI:

```
borrowable_eth, total_debt_eth = get_borrowable_data(lending_pool,
account)
print(fLETS BORROW IT ALL")
erc20_eth_price = getasset_price()
amount_erc20_to_borrow = 1 / erc20_eth_price) * (borrowable_eth *
0.95)
print(fWe are going to borrow {amount_erc20_to_borrow} DAI")
borrowerc20(lending_pool, amount_erc20_to_borrow, account)
borrowable_eth, total_debt_eth = getborrowable_data(lending_pool,
account)
```

The above process includes:

1. Getting the data of how much we can borrow based on how much we have deposited and how much debt we have.
2. Getting the price of the asset we are going to borrow
3. Calculating the amount that we are going to borrow (it is multiplied by 0.95 as we can only borrow 95 percent of our collateral at most.)
4. Borrowing the DAI.

```
def get_borrowable_data(lending_pool, account):
    (total_collateral_eth,
    total_debt_eth,
    available_borrow_eth,
    current_liquidation_threshold,
    tlv,
    health_factor, ) = lending_pool.getherAccountData(account.address)
    available_borrow_eth =Web3.fromWei(available_borrow_eth, "ether")
    total_collateral_eth =Web3.fromWei(total_collateral_eth, "ether")
    total_debt_eth =Web3.fromWei(total_debt_eth, "ether")
    print(fYou have {total_collateral_eth} worth of ETH deposited.")
    print(fYou have {total_debt_eth} worth of ETH borrowed.")
    print(fYou can borrow {available_borrow_eth} worth of ETH.")
    return float(available_borrow_eth), float(total_debt_eth)
```

The above function retrieves the data related to how much we can borrow based on our account and the lending pool we have deposited our money.

```
def borrow_erc20(lending_pool, amount, account, erc20_address=None):
    erc20_address = (
        erc20_address
    if erc20_address
    else config["networks"][network.show_active()][ "aave_dai_token" ])
    transaction = lending_pool.borrow(
        erc20_address,
        Web3.toWei(amount, "ether"),
        1,
        0,
        account.address,
        {"from": account},
    )
    transaction.wait(1)
    print(f"Congratulations! We have just borrowed {amount}")
```

The above function executes the borrowing.

```
def get_asset_price():
    dai_eth_price_feed = interfaceAggregatorV3Interface(
        config["networks"][network.show_active()][ "dai_eth_price_feed" ])
    latest_price = Web3.fromWei(dai_eth_price_feed.latestRoundData()[1],
        "ether")
    print(f"The DAI/ETH price is {latest_price}")
    return float(latest_price)
```

Using the above function, we get the DAI/ETH pair price.

Brownie_config.yaml

The complete brownie-config.yaml file goes like this:

```
dependencies:
  - smartcontractkit/chainlink-brownie-contracts@.4.0
  - aave/protocol-v2@.0.1
compiler:
  solc:
    remappings:
```

```
"@chainlink=smartcontractkit/chainlink-brownie-
"@aave=aave/protocol-v2@1.0.1"
contracts@0.4.0'
autofetch_sources: True
dotenv: .env
verify: False
networks:
  default:mainnet-fork
  mainnet-fork:
    lending_pool_addresses_provider:
      "0xB53C1a33016B2DC2fF3653530bfF1848a515c8c5"
      weth_token:"0xc02aaa39b223fe8d0a0e5c4f27ead9083c756cc2"
      link_token:"0x514910771af9ca656af840dff83e8264ecf986ca"
      aave_link_token:"0x514910771af9ca656af840dff83e8264ecf986ca"
      aave_dai_token:"0x6b175474e89094c44da98b954eedeac495271d0f"
      dai_token:"0x6b175474e89094c44da98b954eedeac495271d0f"
      link_eth_price_feed:"0xDC530D9457755926550b59e8ECcdaE7624181557"
      dai_eth_price_feed:"0x773616E4d11A78F511299002da57A0a94577F1f4"
  kovan:
    vrf_coordinator:"0xdD3782915140c8f3b190B5D67eAc6dc5760C46E9"
    aave_link_token:"0xAD5ce863aE3E4E9394Ab43d4ba0D80f419F61789"
    aave_dai_token:"0xFF795577d9AC8bD7D90Ee22b6C1703490b6512FD"
    link_token:"0xa36085F69e2889c224210F603D836748e7dC0088"
    keyhash:
      "0x6c3699283bda56ad74f6b855546325b68d482e983852a7a82979cc4807b641f4"
      fee1000000000000000000
      oracle:"0x2f90A6D021db21e1B2A077c5a37B3C7E75D15b7e"
      jobId:"29fa9aa13bf1468788b7cc4a500a45b8"
      eth_usd_price_feed:"0x9326BFA02ADD2366b30bacB125260Af641031331"
      link_eth_price_feed:"0x3Af8C569ab77af5230596Acf0E8c2F9351d24C38"
      dai_eth_price_feed:"0x22B58f1EbEDfCA50feF632bd73368b2FdA96D541"

    lending_pool_addresses_provider:
      "0x88757f2f99175387ab4c6a4b3067c77a695b0349"
      weth_token:"0xd0a1e359811322d97991e03f863a0c30c2cf029c"
  wallets:
    from_key:${PRIVATE_KEY}
```

Deploying the Contracts

Now, we can finally borrow our desired asset:

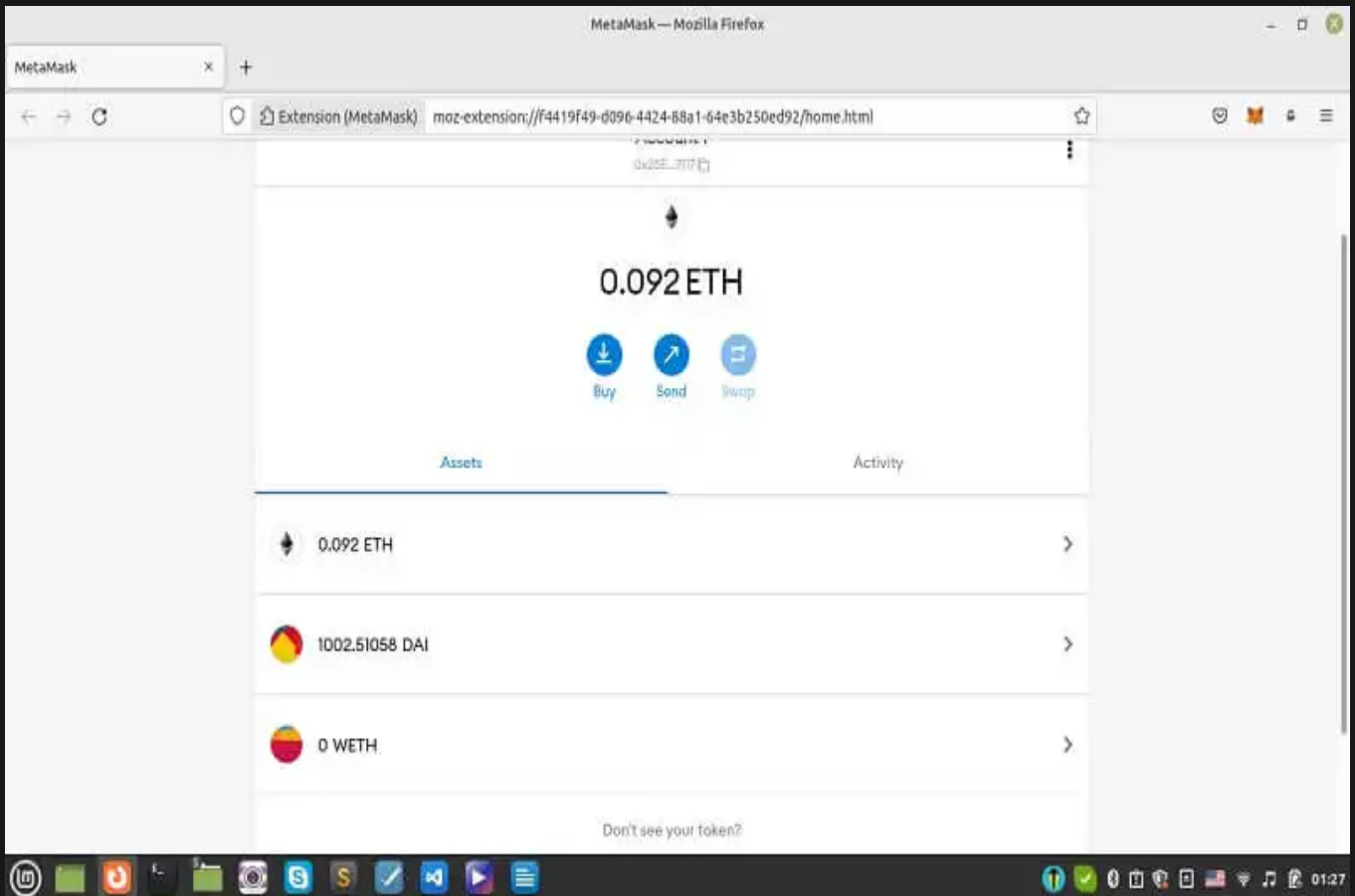
```
brownie run scripts/aave_borrow.py --network kovan
```

Result:

```
Brownie v1.18.1 - Python development framework for
EthereumAaveBrownieProject is the active project.Running
'scripts/aave_borrow.py::main'... Approving ERC20... Transaction sent:
0x0612e2c08897577cedef93950fb35d4e8710c29032d6ae5dcb0f202a27d9f3b2 Gas
```

```
price: 2.500000007 gwei Gas limit: 50695 Nonce: 15 WETH9_.approve
confirmed Block: 30943344 Gas used: 46087 (90.91%)WETH9_.approve
confirmed Block: 30943344 Gas used: 46087 (90.91%)Approved!
Depositing... Transaction sent:
0xf2fe4bb2e82189d2173cf7b147c24af47fa172061a0ac7c37a434ddc68bd984f
Fetching source of 0x2646FcF7F0AbB1ff279ED9845AdE04019C907EBE from
api-kovan.etherscan.io... Gas price: 2.500000007 gwei Gas limit:
207906 Nonce: 16 Transaction confirmed Block: 30943347 Gas used:
176916 (85.09%)Deposited! You have 0.400410656100777314 worth of ETH
deposited. You have 1.1287963056E-8 worth of ETH borrowed. You can
borrow 0.320328513592658795 worth of ETH. LETS BORROW IT ALL The
DAI/ETH price is 0.00030355 We are going to borrow 1002.5105844606354
DAI Transaction sent:
0xd1d8b7f23e2ea35eb12dc30f5142d5614068a652e77fa811c6830dde766f1f26
Fetching source of 0x2646FcF7F0AbB1ff279ED9845AdE04019C907EBE from
api-kovan.etherscan.io... Gas price: 2.500000007 gwei Gas limit:
353287 Nonce: 17 Transaction confirmed Block: 30943350 Gas used:
316618 (89.62%)This transaction already has 2 confirmations.
Congratulations! We have just borrowed 1002.5105844606354 You have
0.400410804230975243 worth of ETH deposited. You have
0.304312101427770057 worth of ETH borrowed. You can borrow
0.016016541957010137 worth of ETH.
```

And as you can see we have got nearly 1002 DAI tokens as a result of borrowing from the Aave pool. Notice that the amount of deposited ETH is 0.4 which is the result of multiple times we have tested the deposit function and also a little more than 0.4 is because of the yield that Aave pool has considered for us as a result of depositing our ETH. In the next part, we are going to repay all of the borrowed DAI.



In this tutorial, after depositing WETH and borrowing Dai, it is time to repay back the funds we have deposited. To do this we will modify the `deploy.py` to completely do the 3 processes of swapping ETH for WETH, depositing WETH, borrowing DAI, and repaying the funds. As in the recent steps we hadn't completely done the 4 tasks and as a result, we hadn't repaid back the funds, we need to manually repay the remaining funds.

Modifying the `deploy.py`:

The final step of interacting with the Aave lending pool is to repay the DAI that we have borrowed. The complete steps of depositing, borrowing, and repaying back are included in this main function with the other functions we created in the recent parts:

```
def main():
    account = get_account()
    erc20_address = config["networks"][network.show_active()]["weth_token"]
    if network.show_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_pool.deposit(erc20_address, amount, account.address,0, {
"from": account})
print("Deposited!")
borrowable_eth, total_debt_eth = get_borrowable_data(lending_pool,
account)
print("LETS BORROW IT ALL")
erc20_eth_price = get_asset_price()
amount_erc20_to_borrow = 1 / erc20_eth_price * (borrowable_eth *
0.95)
print(f>We are going to borrow {amount_erc20_to_borrow} DAI")
borrowerc20(lending_pool, amount_erc20_to_borrow, account)
borrowable_eth, total_debt_eth = get_borrowable_data(lending_pool,
account)
```

We continue the def main() with the following:

```
repayall(amount_erc20_to_borrow, lending_pool, account)
get_borrowable_data(lending_pool, account)
```

And also, create a new function called repay_all:

```
def repay_all(amount, lending_pool, account):
approve_erc20(Web3.toWei(amount, "ether"),
lending_pool,
config["networks"][network.show_active()][ "aave_dai_token" ],
account,
)
tx = lending_pool.repay(
config["networks"][network.show_active()][ "aave_dai_token" ],
Web3Wei(amount, "ether"),
1,
account.address,
{"from": account},
)
txwait(1)
print("Repaid!")
```

Deploying on the Kovan network:

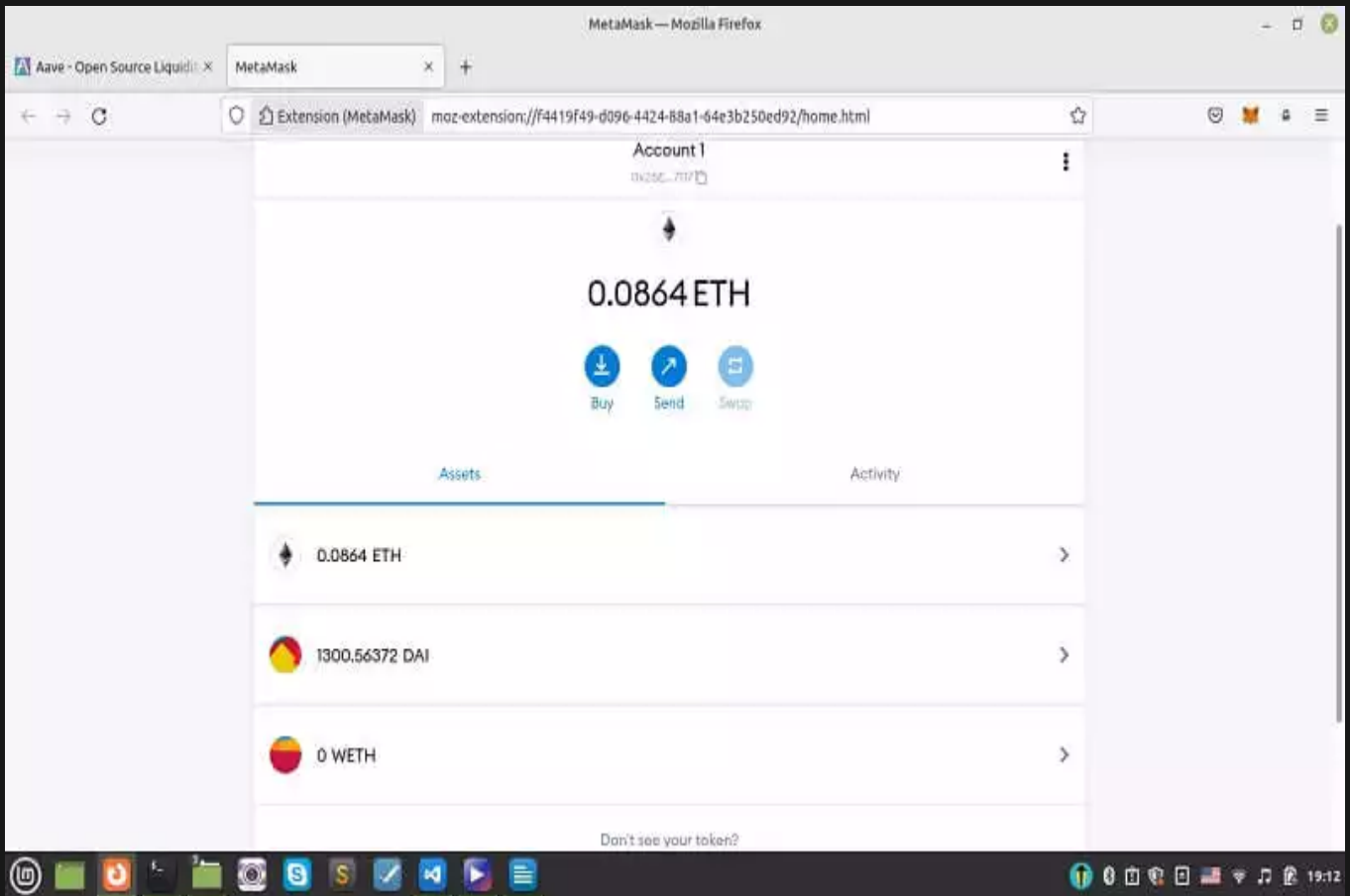
Now, it is time to test the whole process:

```
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:
0x607f92b4795a1b344f27eb517b14b382b3af62060861bcb7e17fe56f2e49dbf4 Gas
price: 2.500000007 gwei Gas limit: 50695 Nonce: 24 WETH9_.approve
confirmed Block: 30944173 Gas used: 46087 (90.91%)WETH9_.approve
confirmed Block: 30944173 Gas used: 46087 (90.91%)Approved!
Depositing... Transaction sent:
0x8b19aae38d08356a19a336746530be83fb621e64c566d8777382f04dd97feb61Fetching
source of 0x2646FcF7F0AbB1ff279ED9845AdE04019C907EBE from api-
kovan.etherscan.io... Gas price: 2.500000007 gwei Gas limit: 207906
Nonce: 25 Transaction confirmed Block: 30944175 Gas used: 176916
(85.09%)Deposited! You have 0.600462332371017026 worth of ETH
deposited. You have 0.395594815443779104 worth of ETH borrowed. You
can borrow 0.084775050453034517 worth of ETH. LETS BORROW IT ALL The
DAI/ETH price is 0.00030417 We are going to borrow 264.7739682755788
DAI Transaction sent:
0x114bf44dlee6ca2490dbb32276e9f7abb91078c2b157c42edede91fbc331024d
Fetching source of 0x2646FcF7F0AbB1ff279ED9845AdE04019C907EBE from
api-kovan.etherscan.io... Gas price: 2.500000007 gwei Gas limit:
330966 Nonce: 26 Transaction confirmed Block: 30944177 Gas used:
299518 (90.50%)This transaction already has 2 confirmations.
Congratulations! We have just borrowed 264.7739682755788 You have
0.600462628542463152 worth of ETH deposited. You have
0.476131125542422592 worth of ETH borrowed. You can borrow
0.00423897729154793 worth of ETH. Approving ERC20... Transaction sent:
0x442a4ae4ee05339a7081fd4c91d231e71d9f30dfc489cc1ba511e3c64143587a Gas
price: 2.500000007 gwei Gas limit: 31988 Nonce: 27 Transaction
confirmed Block: 30944180 Gas used: 29080 (90.91%)Transaction
confirmed Block: 30944180 Gas used: 29080 (90.91%)Approved!
Transaction sent:
0x20529c85bc1852bce18341ed3404094ba1af3db58f81ade3145edea7afefb330
Fetching source of 0x2646FcF7F0AbB1ff279ED9845AdE04019C907EBE from
api-kovan.etherscan.io... Gas price: 2.500000007 gwei Gas limit:
242655 Nonce: 28 Transaction confirmed Block: 30944181 Gas used:
212817 (87.70%)Transaction confirmed Block: 30944181 Gas used: 212817
(87.70%)Repaid! You have 0.60046299875677081 worth of ETH deposited.
You have 0.395594844442992547 worth of ETH borrowed. You can borrow
0.084775554562424101 worth of ETH.
```

As you can see, we have successfully deposited WETH, borrowed some DAI, and repaid what we have borrowed. Now let's check our Metamask wallet.



Repaying all funds:

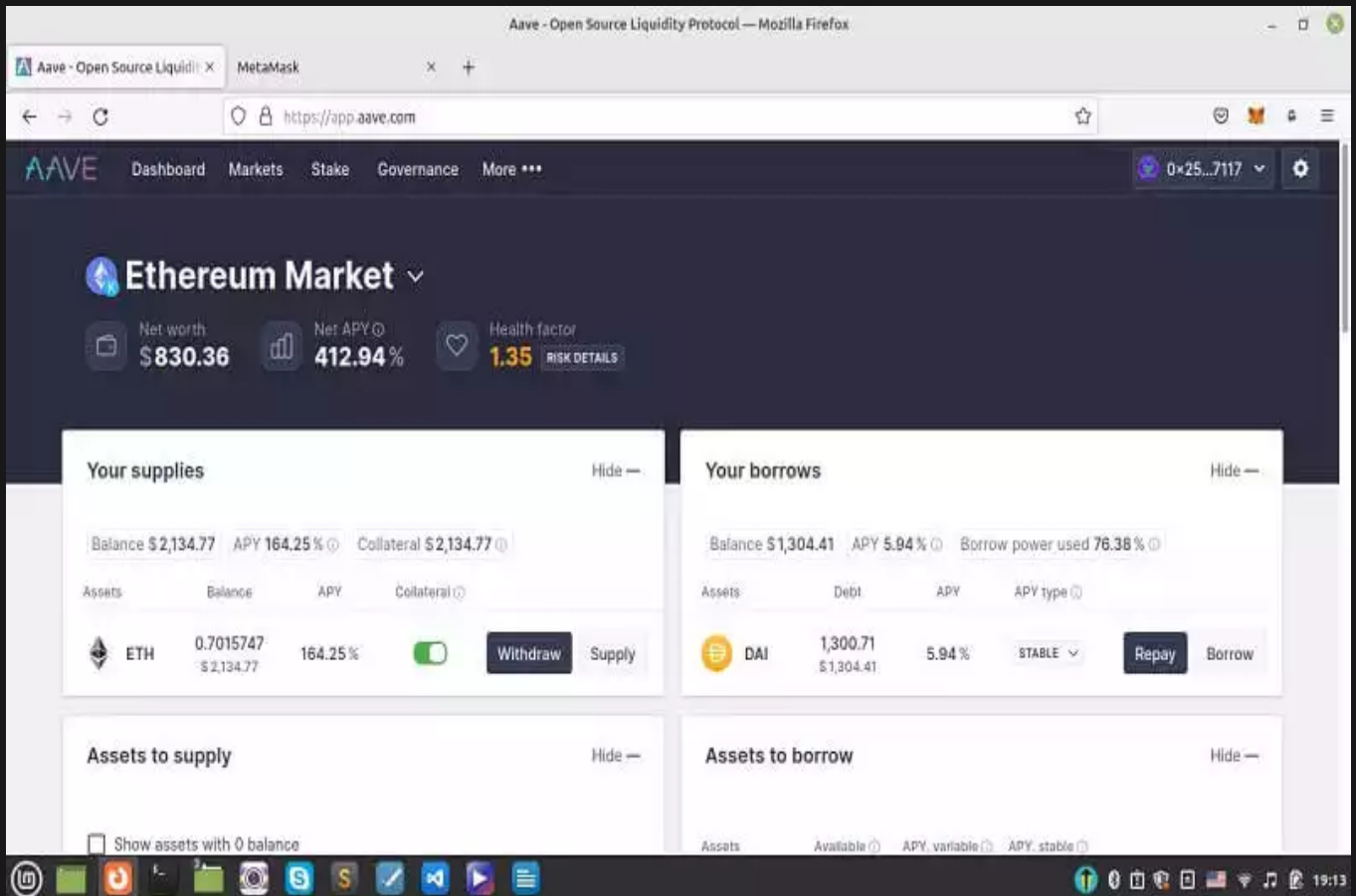
You might wonder why we still have some DAI in it. The reason is that, in our recent code run, we didn't have to repay. As a result, we still have some of the DAIs that we haven't repaid back. To do so, change the main function to:

```
def main():
    account = get_account()
    erc20_address = config["networks"][network.show_active()]["weth_token"]
    if network.show_active() in ["mainnet-fork"]:
        get_weth(account=account)
        lending_pool = get_lending_pool()
        approve_erc20(amount, lending_pool.address, erc20_address, account)
        borrowable_eth, total_debt_eth = get_borrowable_data(lending_pool, account)
        amount_erc20_to_repay = # enter the remaining to repay
```

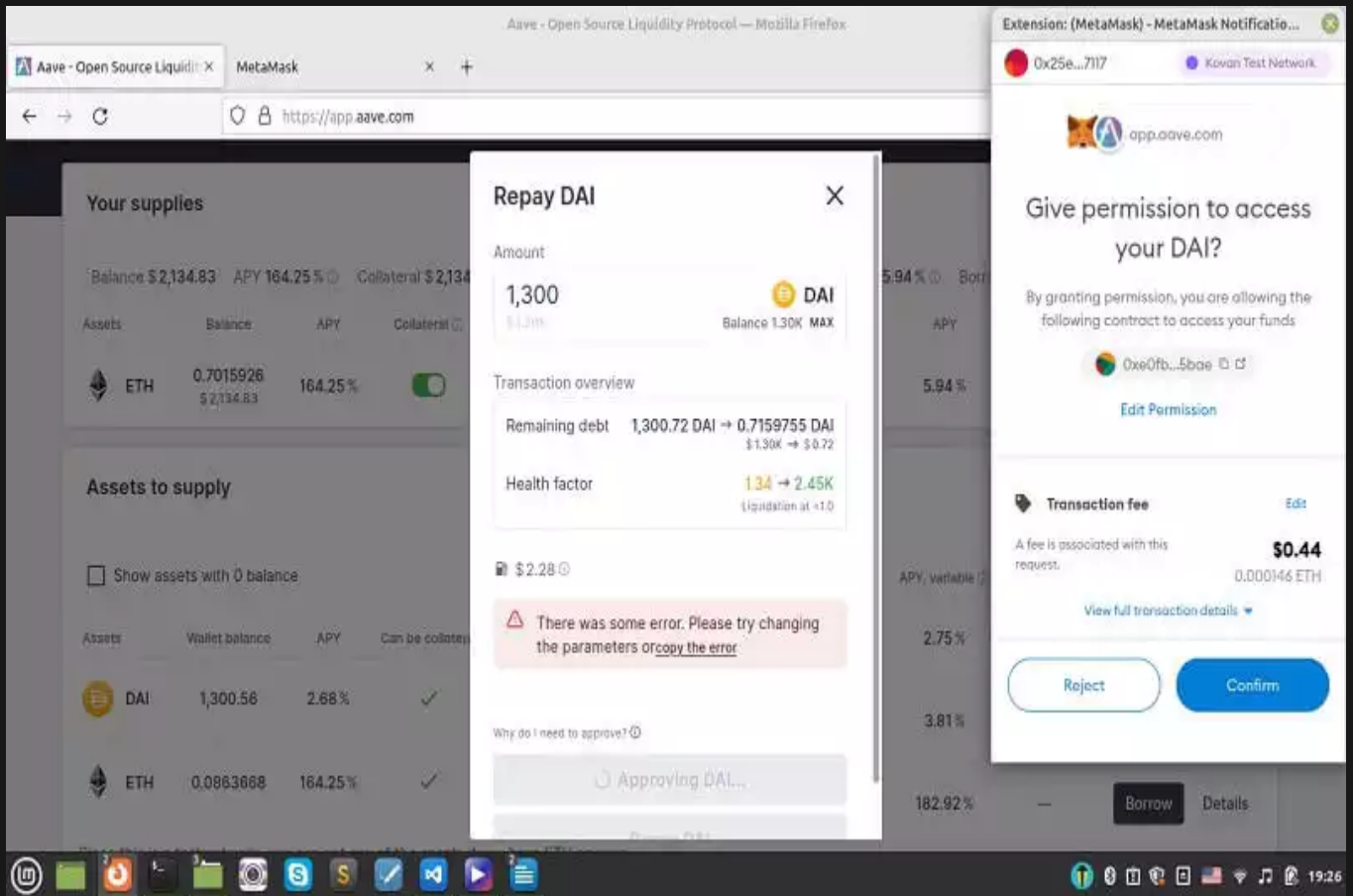
We should determine the amount we want to repay in the amount_erc20_to_repay.

```
repay_all(amount_erc20_to_repay, lending_pool, account)  
get_borrowable_data(lending_pool, account)
```

Or instead, you can repay manually using the Aave interface:



Click repay next to DAI and then select max under the DAI icon in the pop-up window, and click approve, after that confirm the Metamask pop-up:

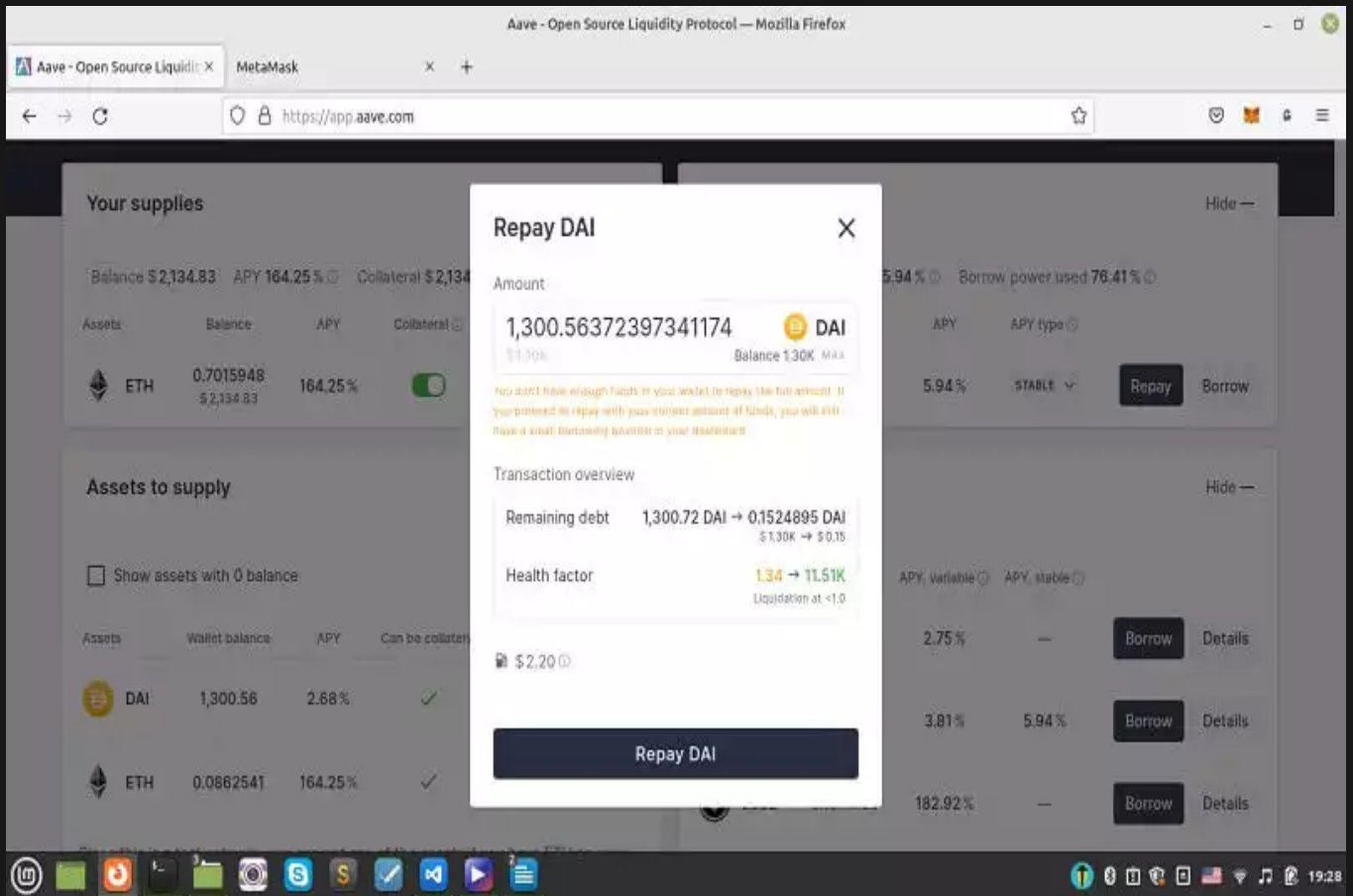


The screenshot displays the Aave application interface with a 'Repay DAI' modal open. The modal shows a repayment amount of 1,300 DAI and a transaction overview. An error message is displayed: 'There was some error. Please try changing the parameters or copy the error'. A MetaMask confirmation overlay is also visible, asking for permission to access DAI funds. The background shows the 'Your supplies' and 'Assets to supply' sections of the app.

| Assets | Balance | APY | Collateral |
|--------|-----------|---------|------------|
| ETH | 0.7015926 | 164.25% | 2,134.83 |

| Assets | Wallet balance | APY | Can be collateral |
|--------|----------------|---------|-------------------|
| DAI | 1,300.56 | 2.68% | ✓ |
| ETH | 0.0863668 | 164.25% | ✓ |

Now, you can repay DAI:

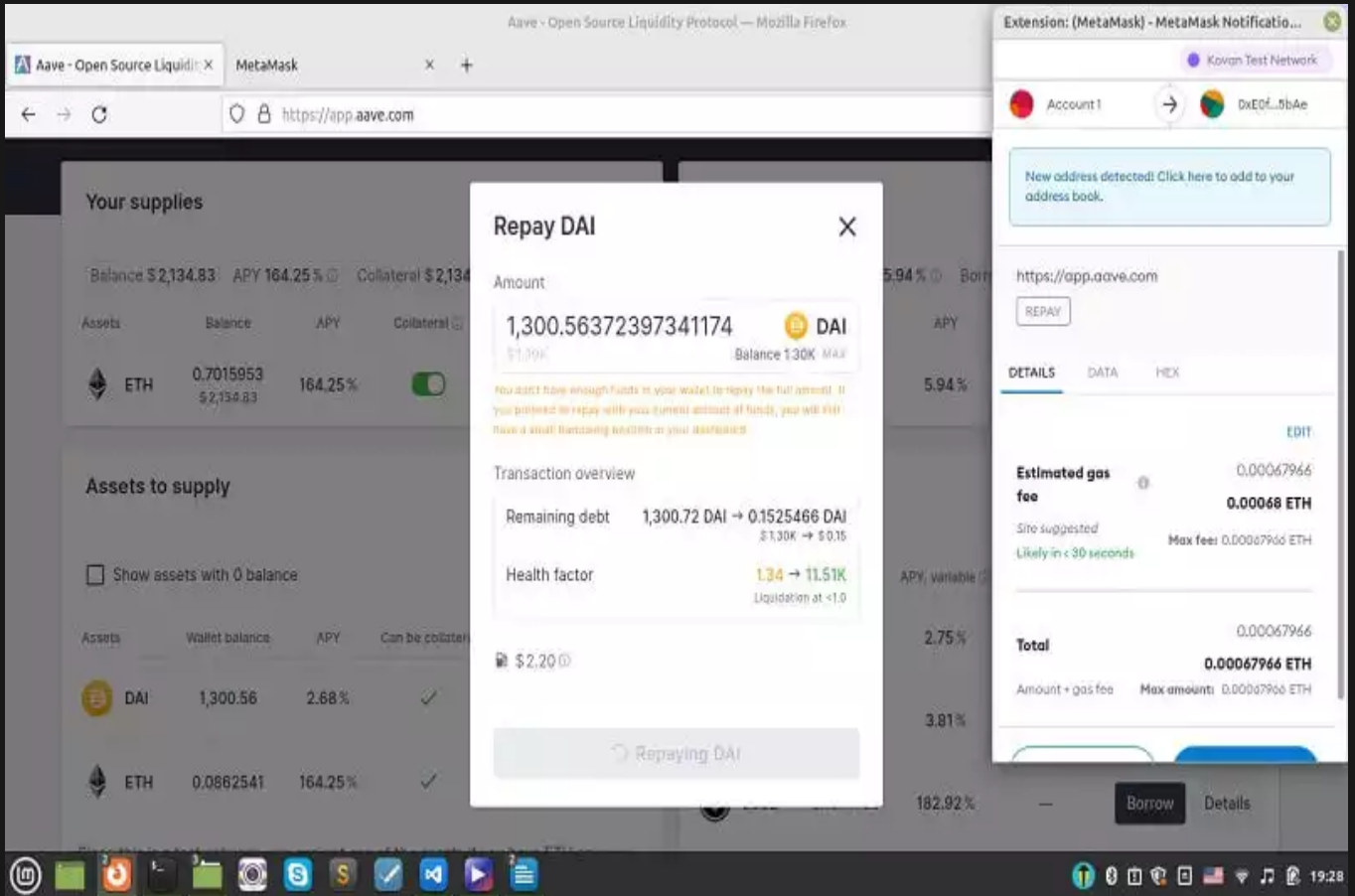


The screenshot shows the Aave web application interface with a 'Repay DAI' modal window open. The modal displays the following information:

- Amount:** 1,300.56372397341174 DAI (Balance 1.30K MAX)
- Transaction overview:**
 - Remaining debt: 1,300.72 DAI → 0.1524895 DAI (\$1,30K → \$0.15)
 - Health factor: 1.34 → 11.51K (Liquidation at <1.0)
- Gas fee:** \$2.20
- Action:** A large 'Repay DAI' button is visible at the bottom of the modal.

The background interface shows the 'Your supplies' section with a balance of \$2,134.83 and APY of 164.25%. The 'Assets to supply' section lists DAI and ETH assets with their respective wallet balances and APYs.

Confirm Metamask pop-up again:

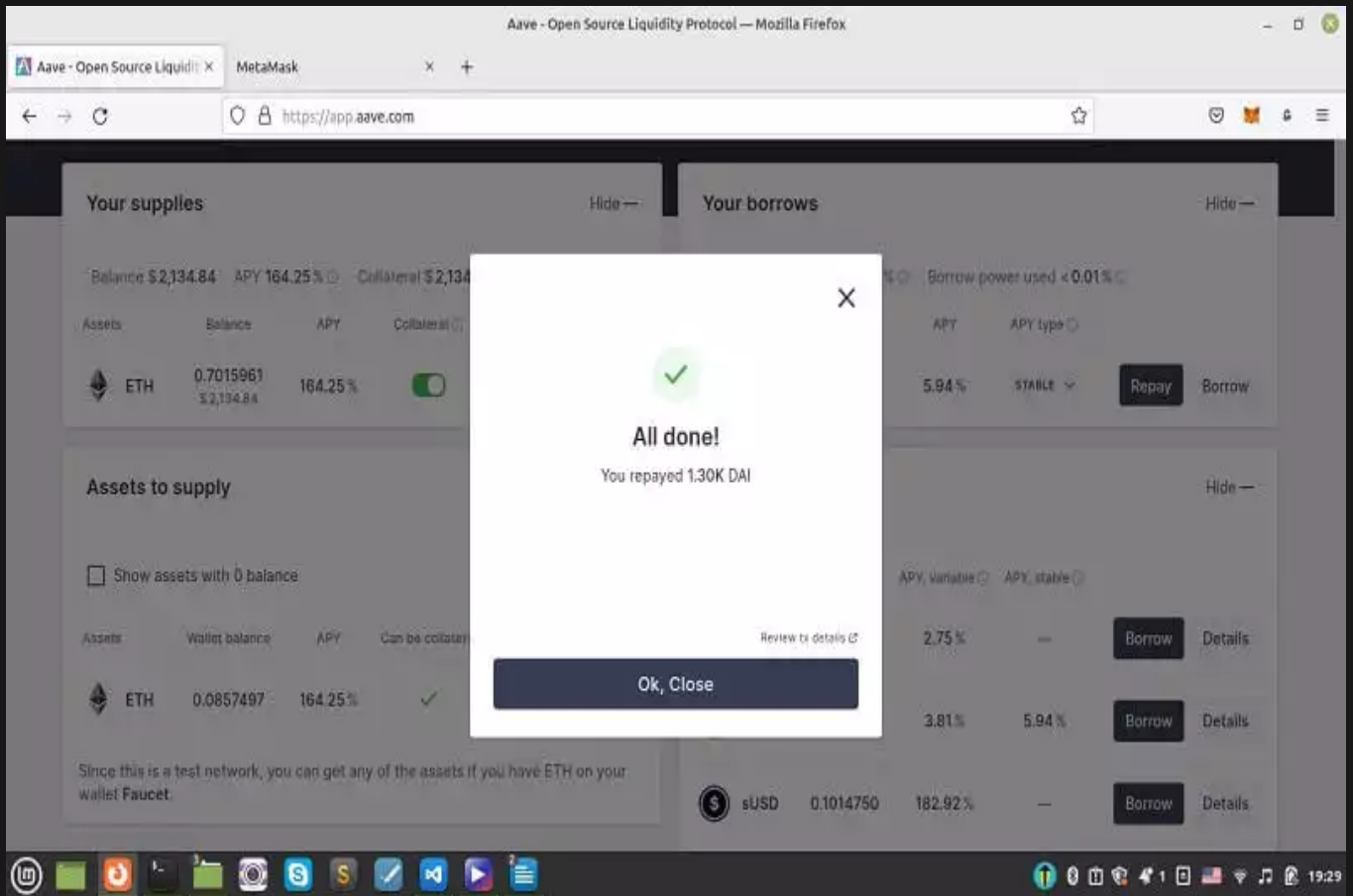


The screenshot displays the Aave web application interface with a 'Repay DAI' modal window open. The modal shows a transaction amount of 1,300.56372397341174 DAI, with a balance of 1.30K and a maximum value of 1.30K. It also displays a transaction overview with a remaining debt of 1,300.72 DAI and a health factor of 1.34. A 'Repaying DAI' button is visible at the bottom of the modal.

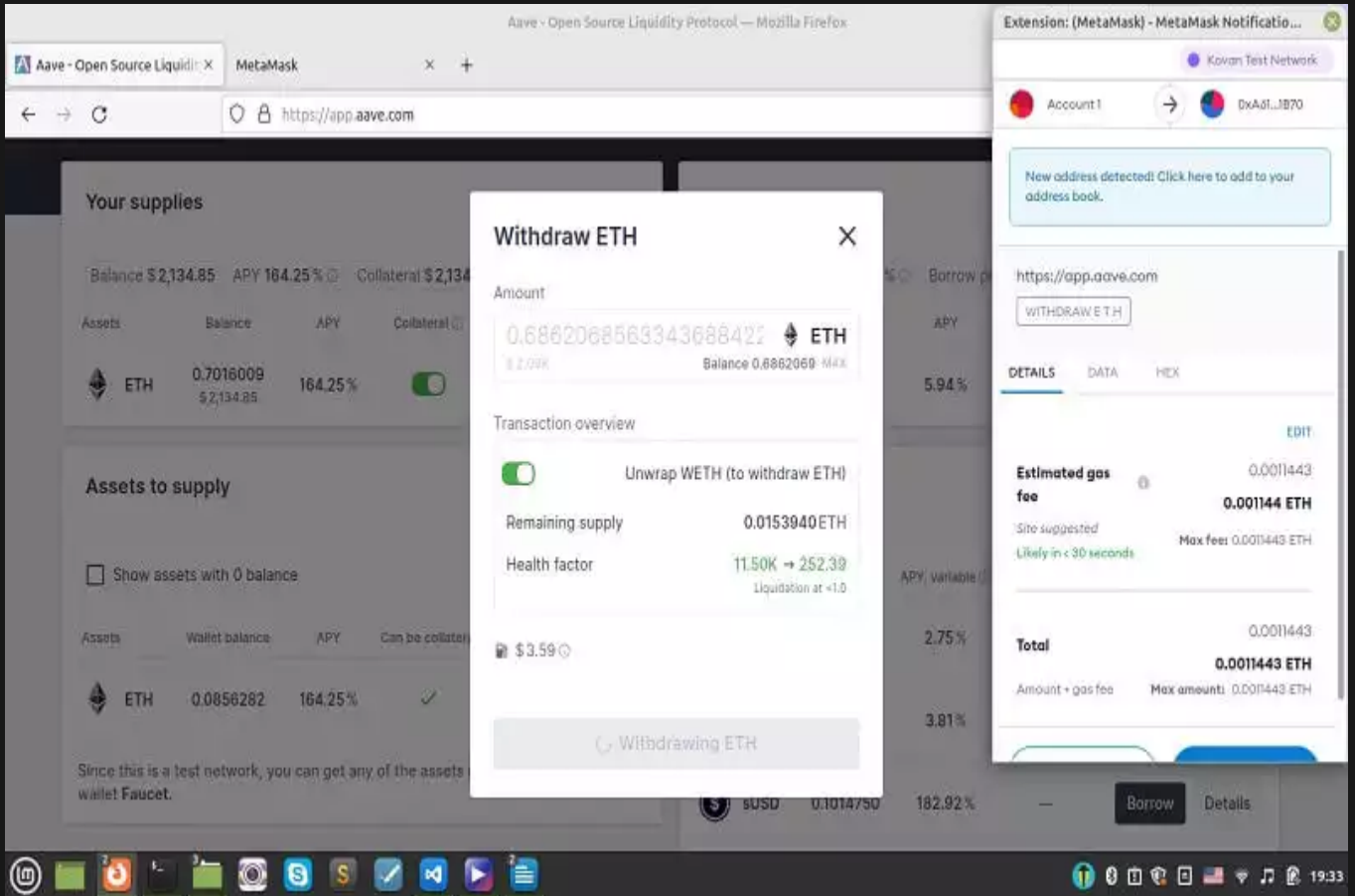
In the background, the 'Your supplies' section shows a balance of \$2,134.83 and an APY of 164.25%. The 'Assets to supply' section includes a table with columns for Assets, Wallet balance, APY, and Can be collateral:

| Assets | Wallet balance | APY | Can be collateral |
|--------|----------------|---------|-------------------|
| DAI | 1,300.56 | 2.68% | ✓ |
| ETH | 0.0862541 | 164.25% | ✓ |

A MetaMask extension notification is visible on the right side of the screen, showing a 'New address detected' message and a 'REPAY' button. The notification also displays the URL 'https://app.aave.com' and estimated gas fees of 0.00067966 ETH.



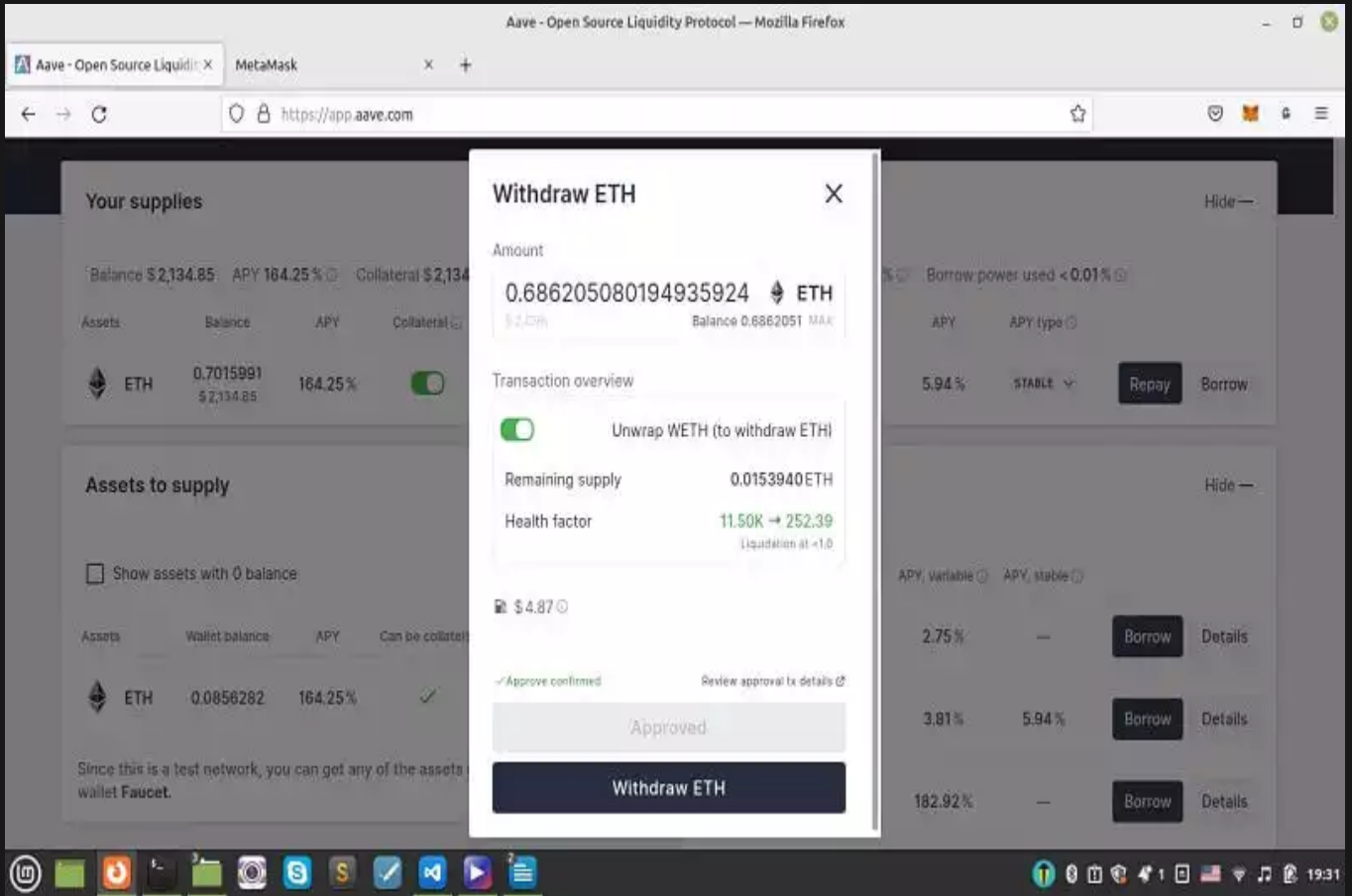
You can also withdraw by following the steps in the below photos. Notice that for both repay and withdraw you need to get approval from your Metamask wallet and click confirm and then repay or withdraw.



The screenshot displays the Aave web application interface with a 'Withdraw ETH' modal open. The modal shows the amount to be withdrawn as 0.6862068563343688422 ETH, with a balance of 0.6862069 ETH. It also displays transaction details such as 'Unwrap WETH (to withdraw ETH)', 'Remaining supply: 0.0153940 ETH', and 'Health factor: 11.50K → 252.30'. A gas fee of \$3.59 is shown at the bottom of the modal.

In the background, the Aave interface shows 'Your supplies' with a balance of \$2,134.85 and APY of 184.25%. The 'Assets to supply' section includes a table with columns for Assets, Wallet balance, APY, and Can be collateralized. The table shows one asset: ETH with a wallet balance of 0.0856282 and an APY of 164.25%.

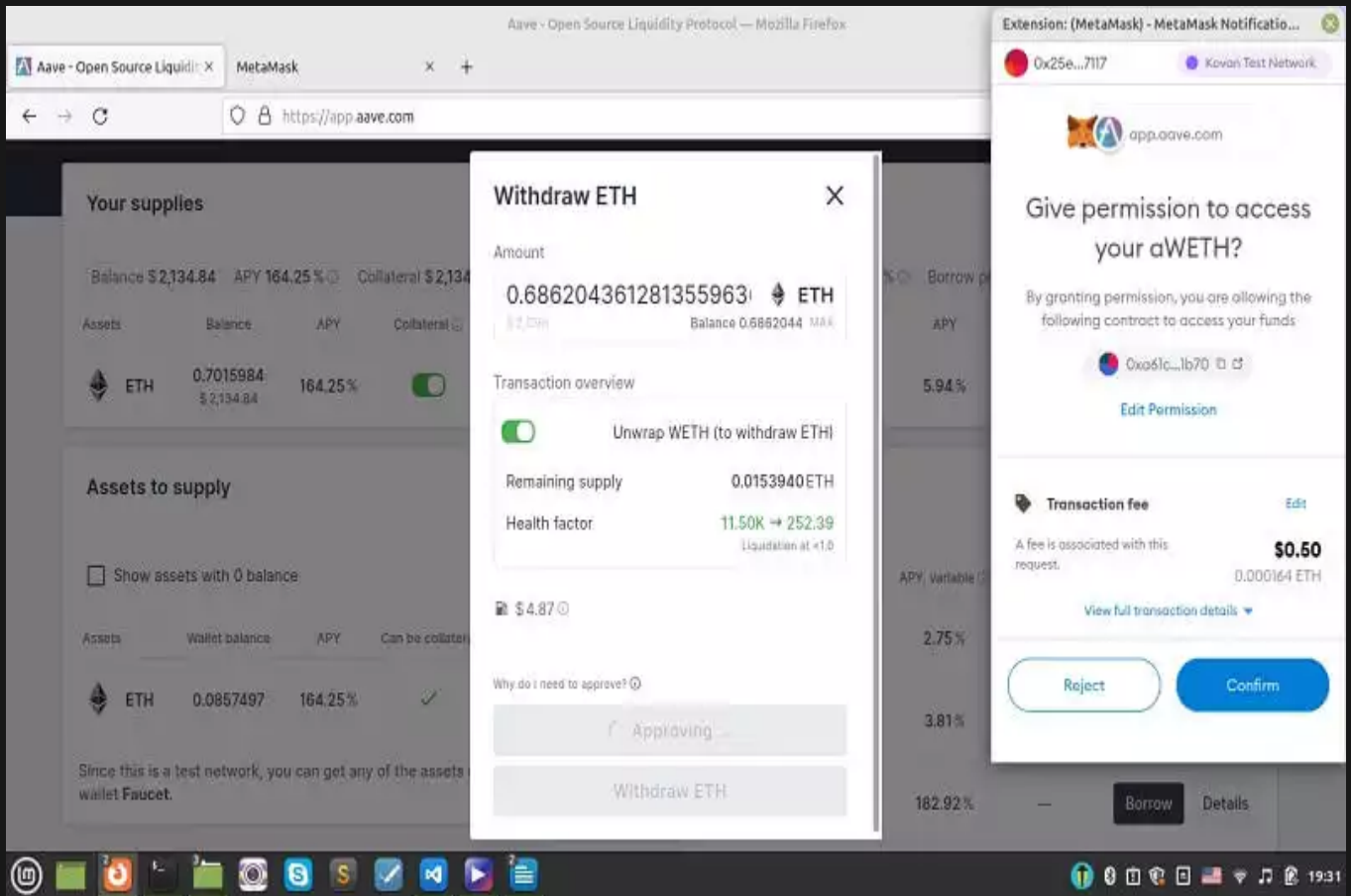
On the right side, a MetaMask notification is visible, showing a 'New address detected' message and a 'WITHDRAW ETH' button. The notification also displays the estimated gas fee of 0.0011443 ETH and the total amount of 0.0011443 ETH.



The screenshot displays the Aave web application interface. A modal window titled "Withdraw ETH" is open in the center, showing the amount 0.686205080194935924 ETH. The modal includes a "Transaction overview" section with a toggle for "Unwrap WETH (to withdraw ETH)" and a "Withdraw ETH" button. The background shows the "Your supplies" section with a balance of \$2,134.85 and APY of 164.25%, and the "Assets to supply" section with a table of assets including ETH with a balance of 0.7015991 and APY of 164.25%.

| Assets | Balance | APY | Collateral |
|--------|-----------|---------|------------|
| ETH | 0.7015991 | 164.25% | |

| Assets | Wallet balance | APY | Can be collateral |
|--------|----------------|---------|-------------------|
| ETH | 0.0856282 | 164.25% | ✓ |



Conclusion

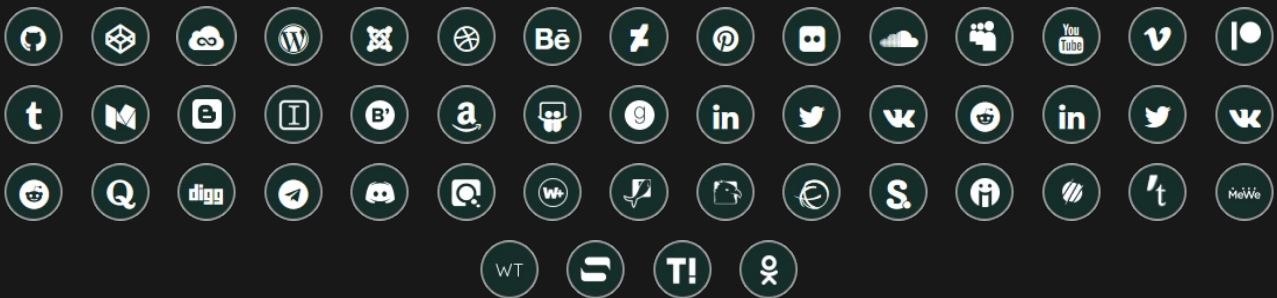
In this article, we have managed to borrow some Dai from the Aave lending pool. To do this, we have added some dependency contracts like LinkTokenInterface.sol and AggregatorV3Interface.sol. We have also modified the deploy.py file to do the borrowing function for us.

In this tutorial, we have managed to do the complete process of the Aave lending pool, meaning that we have swapped ETH for WETH, deposited WETH, borrowed Dai, and finally repaid borrowed Dai using Brownie python tools.

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 beneficial 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](#)