

# Using Tenderly to simulate a transaction

Tenderly is a Smart contract monitoring and alerting tool. With tenderly you can monitor smart contracts on multiple Ethereum networks. Recently, tenderly has come up with a great feature of simulating a transaction on Ethereum network which makes debugging a lot more easier than before.

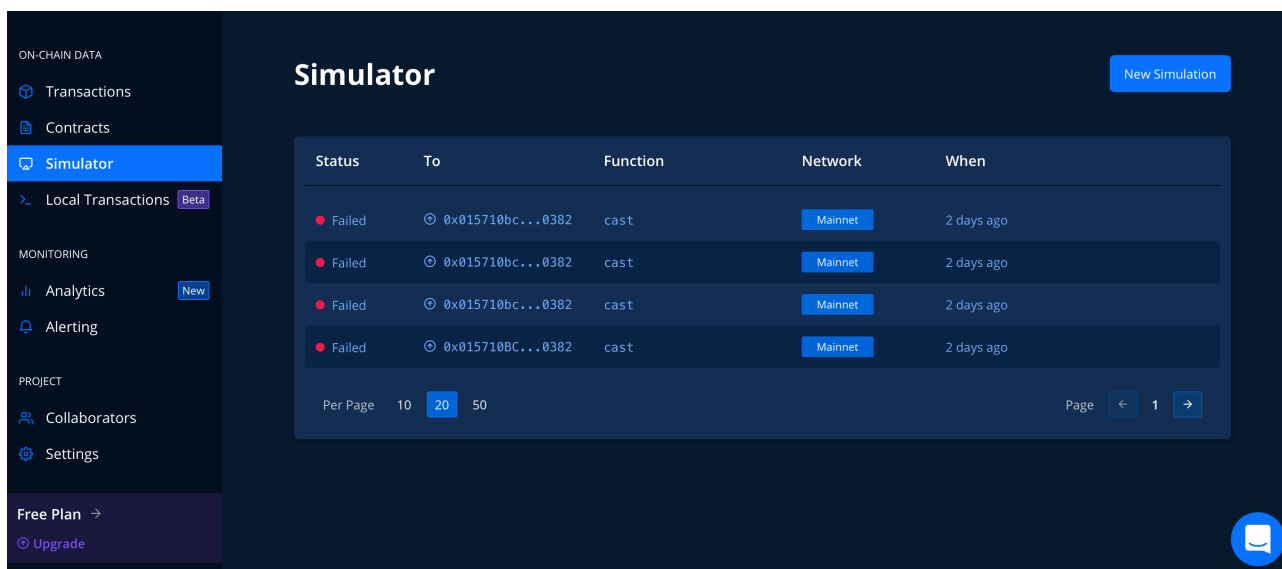
So, today you will learn how you can simulate a transaction on tenderly.

## Step 1:

Create an account on [tenderly](#)

## Step 2:

Select Simulator from the side bar



## Step 3:

Create a new simulation using “New Simulation” button present on the right side of the screen.

Now, We will take a scenario from DSA which will throw an error of “gas required exceeds allowance” which is quite common.

We will try to cast the oasis’ sell spell and maker’s borrow spell with amount 0.2 DAI and slippage 1%.

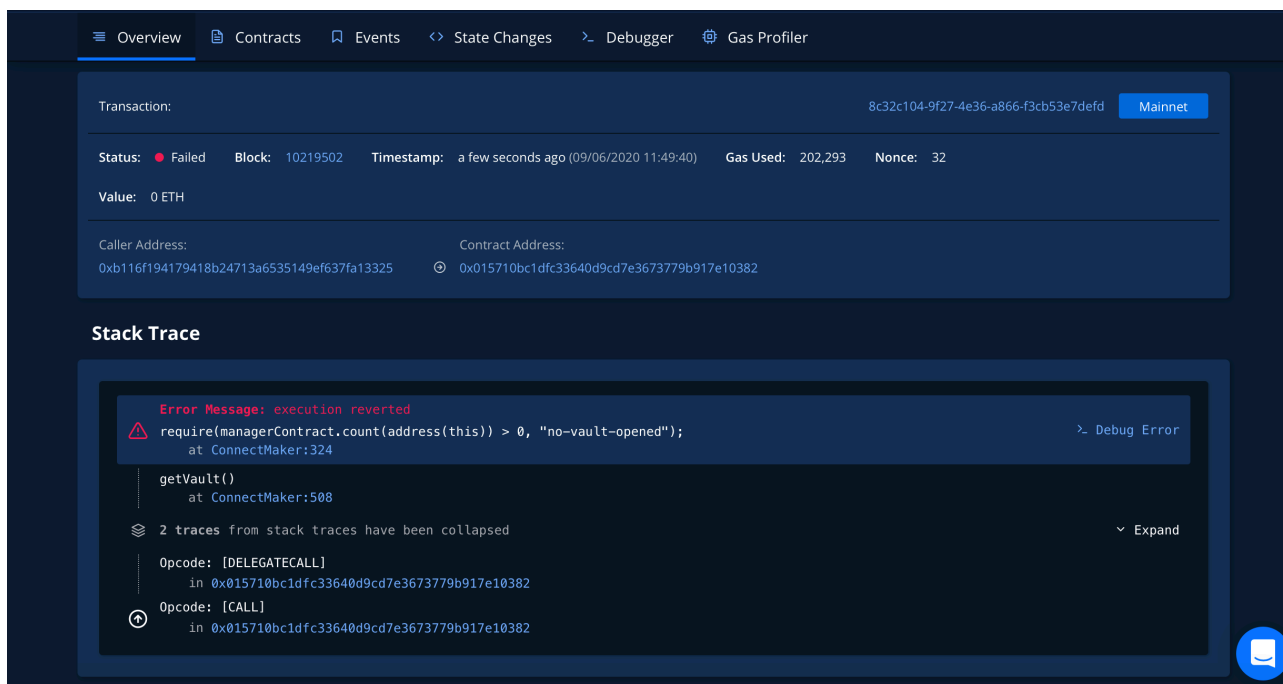
Now, If we will cast the above spells, then it will thrown an error of “gas required exceeds allowance”. But it doesn’t give any clue why it is throwing this error.

So, now we can use the DSA’s `estimateCastGas()` function which provide us the amount of `gasLimit` we need to provide to the spells and in its `Catch` section we will print the error which will give us the data variable in case any error occurs.

Now, after executing the `dsa.estimateCastGas()` function, it will return us error along with the data variable which we will use at tenderly.



Now, When we simulate the transaction it will tell us with the whole call stack and if the transaction will go through or not.



The screenshot shows a debugger interface with a dark theme. At the top, there are navigation tabs: Overview, Contracts, Events, State Changes, Debugger, and Gas Profiler. Below the tabs, a transaction summary is displayed with the following details:

- Transaction: 8c32c104-9f27-4e36-a866-f3cb53e7defd (Mainnet)
- Status: Failed (indicated by a red dot)
- Block: 10219502
- Timestamp: a few seconds ago (09/06/2020 11:49:40)
- Gas Used: 202,293
- Nonce: 32
- Value: 0 ETH

Below the transaction details, the Caller Address and Contract Address are shown:

- Caller Address: 0xb116f194179418b24713a6535149ef637fa13325
- Contract Address: 0x015710bc1dfc33640d9cd7e3673779b917e10382

The main section is titled "Stack Trace" and contains an error message:

```
Error Message: execution reverted
require(managerContract.count(address(this)) > 0, "no-vault-opened");
  at ConnectMaker:324
```

Below the error message, the stack trace is shown:

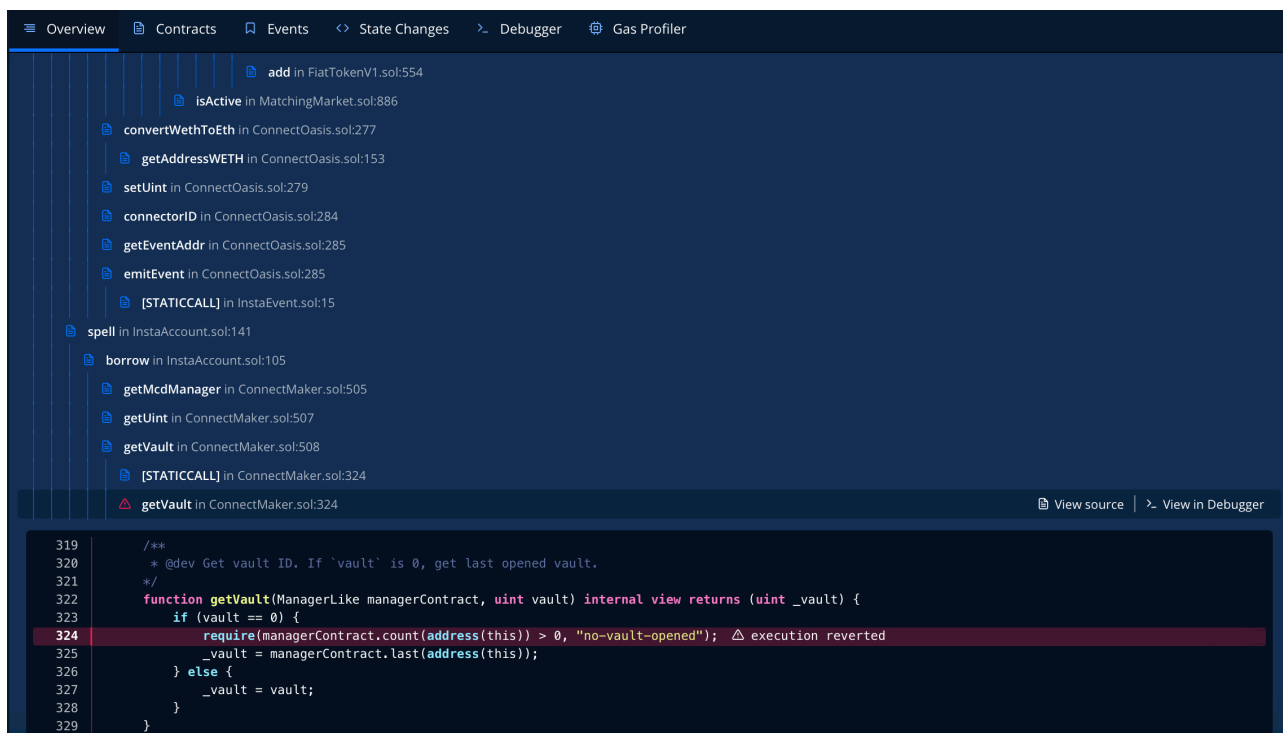
```
getVault()
  at ConnectMaker:508
```

There are 2 traces from stack traces that have been collapsed. The stack trace includes the following opcodes:

- Opcodes: [DELEGATECALL] in 0x015710bc1dfc33640d9cd7e3673779b917e10382
- Opcodes: [CALL] in 0x015710bc1dfc33640d9cd7e3673779b917e10382

In our case, it is showing that the transaction has failed along with an error “execution reverted”.

But this time the error is quite understandable as it is providing us the condition which caused the error which in our case is “*require(managerContract.count(address(this)) > 0, "no-vault-opened");*”. Now, we can understand the error and work on solving it.

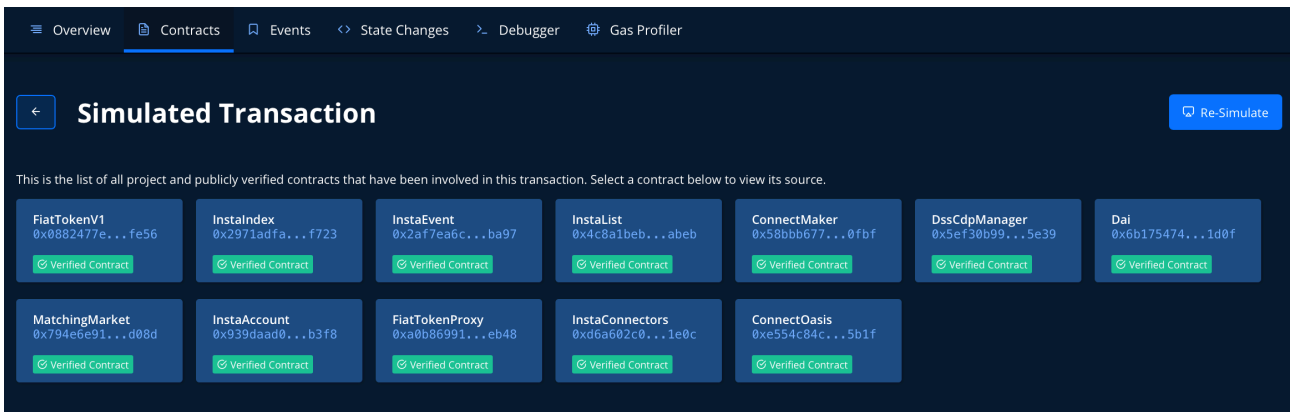


The screenshot shows the same debugger interface, but now displaying the source code of the `getVault` function. The function is located in `ConnectMaker.sol` at line 324. The code is as follows:

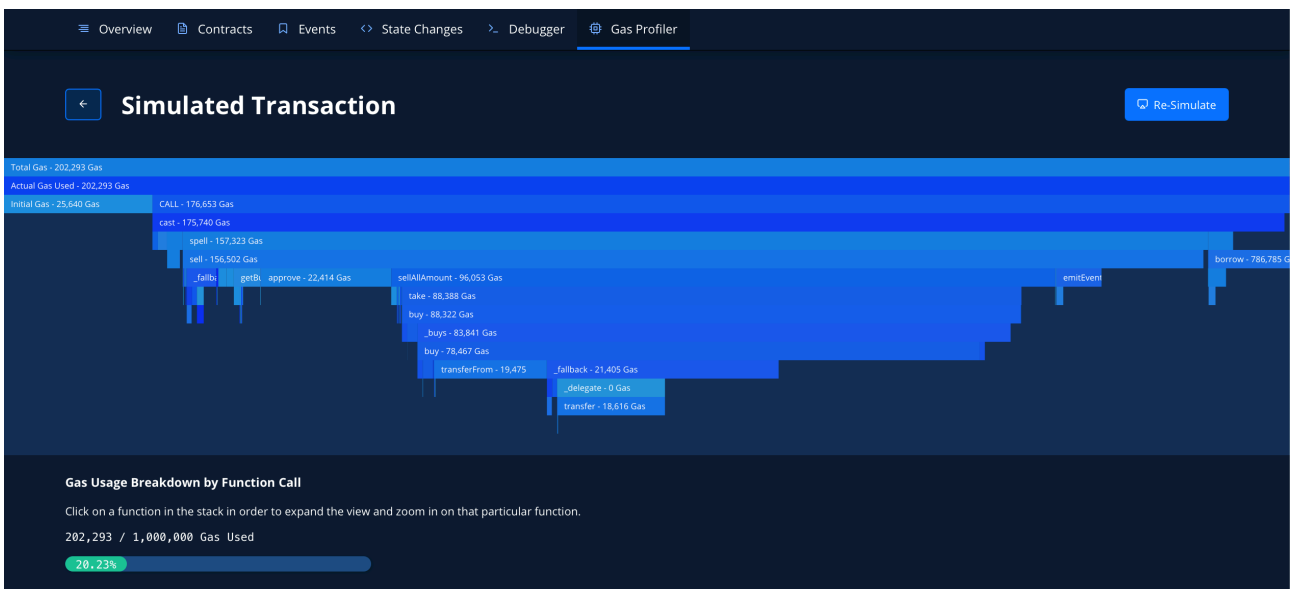
```
319  /**
320   * @dev Get vault ID. If 'vault' is 0, get last opened vault.
321   */
322  function getVault(ManagerLike managerContract, uint vault) internal view returns (uint _vault) {
323      if (vault == 0) {
324          require(managerContract.count(address(this)) > 0, "no-vault-opened");
325          _vault = managerContract.last(address(this));
326      } else {
327          _vault = vault;
328      }
329  }
```

The error message from the previous screenshot is highlighted in red on line 324, indicating that the `require` statement failed because the condition `managerContract.count(address(this)) > 0` was not met.

You can also take a look at the whole Stack Trace which your transaction went through.



If you want to know about the contracts that were involved in the transaction you can navigate to the “Contracts” section and take a look.



There is also a feature of “Gas Profiler” which provides you with a gas usage breakdown by the function call.

Using these steps you can simulate a transaction on tenderly and debug your transaction.