-
  - getBlockWithTransactions - SDK
-
  - estimateGas - SDK
-
  - getBlockNumber - SDK
-
  - getCode - SDK
-
  - getFeeData - SDK
-
  - getLogs - SDK
-
  - getNftMetadataBatch - SDK
-
  - getTokensForOwner - SDK
-
  - getStorageAt - SDK
-
  - getTokenBalances - SDK
-
  - getTransactionCount - SDK
-
  - getTokenMetadata - SDK
-
  - getTransactionReceipt - SDK
-
  - send - SDK
-
  - getTransactionReceipts - SDK
-
  - getTransaction - SDK
-
  - isContractAddress - SDK
-
  - getNftMetadata - SDK
-
  - getNftSales - SDK
-
  - cancelPrivateTransaction - SDK
-
  - sendPrivateTransaction - SDK
-
  - traceTransaction - SDK
-
  - simulateExecutionBundle - SDK
-
  - simulateExecution - SDK
-
  - getMaxPriorityFeePerGas - SDK
-
  - simulateAssetChangesBundle - SDK
-
  - estimateGas - SDK
-
  - simulateAssetChanges - SDK
-
  - traceBlock - SDK
-
  - waitForTransaction - SDK
-
  - traceCall - SDK
-
  - sendTransaction - SDK
-
  - updateWebhook - SDK
-
  - refreshNftMetadata -SDK
-
  - createWebhook - SDK
-
  - getNftFilters - SDK
-
  - getAddresses - SDK
-
  - summarizeNftAttributes - SDK
-
  - deleteWebhook - SDK
-
  - searchContractMetadata - SDK
-
  - getAllWebhooks - SDK
-
  - getNftsForOwnerIterator - SDK
-
  - getNftsForContractIterator -SDK
-
  - getContractMetadata - SDK
-
  - getTransfersForOwner - SDK
-
  - getNftsForContract - SDK

Ethereum

- Ethereum API Quickstart
- Ethereum API FAQ
-
  - Ethereum Developer Guide to the Merge
-
  - How to decode an eth_call response
-
  - How do I distinguish between a contract address and a wallet address?
- Ethereum API Endpoints
-
  - eth_blockNumber - Ethereum post
-
  - eth_getBalance - Ethereum post
-

  - eth_getLogs - Ethereum post
-
  - eth_chainId - Ethereum post
-
  - eth_getBlockByNumber - Ethereum post
-
  - eth_accounts - Ethereum post
-
  - eth_feeHistory - Ethereum post
-
  - eth_estimateGas - Ethereum post
-
  - eth_gasPrice - Ethereum post
-
  - eth_getBlockTransactionCountByHash - Ethereum post
-
  - eth_getBlockReceipts - Ethereum post
-
  - eth_getBlockTransactionCountByNumber - Ethereum post
-
  - eth_getCode - Ethereum post
-
  - eth_getProof - Ethereum post
-
  - eth_getStorageAt - Ethereum post
-
  - eth_getTransactionByBlockHashAndIndex - Ethereum post
-
  - eth_getTransactionByHash - Ethereum post
-
  - eth_getTransactionCount - Ethereum post
-
  - eth_getTransactionReceipt - Ethereum post
-
  - eth_getUncleByBlockHashAndIndex - Ethereum post
-
  - eth_getUncleByBlockNumberAndIndex - Ethereum post
-
  - eth_getUncleCountByBlockHash - Ethereum post
-
  - eth_getUncleCountByBlockNumber - Ethereum post
-
  - eth_maxPriorityFeePerGas - Ethereum post
-
  - eth_protocolVersion - Ethereum post
-
  - eth_sendRawTransaction - Ethereum post
-
  - net_listening - Ethereum post
-
  - net_version - Ethereum post
-
  - web3_clientVersion - Ethereum post
-
  - web3_sha3 - Ethereum post
-
  - eth_getTransactionByBlockNumberAndIndex - Ethereum post
-
  - eth_call - Ethereum post
-
  - eth_getBlockByHash - Ethereum post
-
  - eth_createAccessList - Ethereum post
-
  - eth_newFilter - Ethereum post
-
  - eth_getFilterChanges - Ethereum post
-
  - eth_getFilterLogs - Ethereum post
-
  - eth_newBlockFilter - Ethereum post
-
  - eth_newPendingTransactionFilter - Ethereum post
-
  - eth_uninstallFilter - Ethereum post
-
  - eth_subscribe
-
  - eth_unsubscribe

Polygon PoS

- Polygon PoS API Quickstart
-
  - Polygon SDK Examples
- Polygon PoS API FAQ
- Polygon PoS API Endpoints
-
  - bor_getAuthor - Polygon PoS post
-
  - bor_getCurrentProposer - Polygon PoS post
-
  - bor_getCurrentValidators - Polygon PoS post
-
  - bor_getRootHash - Polygon PoS post
-
  - eth_accounts - Polygon PoS post
-
  - eth_call - Polygon PoS post
-
  - eth_chainId - Polygon PoS post
-
  - eth_estimateGas - Polygon PoS post
-
  - eth_gasPrice - Polygon PoS post
-
  - eth_getBalance - Polygon PoS post
-
  - eth_getBlockByHash - Polygon PoS post

  - zkevm_batchNumber - Polygon zkEVM post
-
  - eth_getCode - Polygon zkEVM post
-
  - eth_getFilterChanges - Polygon zkEVM post
-
  - eth_getFilterLogs - Polygon zkEVM post
-
  - zkevm_getBatchByNumber - Polygon zkEVM post
-
  - eth_getLogs - Polygon zkEVM post
-
  - zkevm_getBroadcastURI - Polygon zkEVM post
-
  - eth_getStorageAt - Polygon zkEVM post
-
  - zkevm_isBlockConsolidated - Polygon zkEVM post
-
  - eth_getTransactionByBlockHashAndIndex - Polygon zkEVM post
-
  - zkevm_isBlockVirtualized - Polygon zkEVM post
-
  - eth_getTransactionByBlockNumberAndIndex - Polygon zkEVM post
-
  - zkevm_verifiedBatchNumber - Polygon zkEVM post
-
  - eth_getTransactionByHash - Polygon zkEVM post
-
  - zkevm_virtualBatchNumber - Polygon zkEVM post
-
  - eth_getCompilers - Polygon zkEVM post
-
  - eth_getUncleByBlockHashAndIndex - Polygon zkEVM post
-
  - eth_getUncleByBlockNumberAndIndex - Polygon zkEVM post
-
  - eth_getUncleCountByBlockHash - Polygon zkEVM post
-
  - eth_getUncleCountByBlockNumber - Polygon zkEVM post
-
  - eth_protocolVersion - Polygon zkEVM post
-
  - eth_blockNumber - Polygon zkEVM post
-
  - eth_getTransactionReceipt - Polygon zkEVM post
-
  - zkevm_batchNumberByBlockNumber - Polygon zkEVM post
-
  - zkevm_consolidatedBlockNumber - Polygon zkEVM post
-
  - zkevm_estimateFee API - Polygon zkEVM post
-
  - zkevm_estimateGasPrice API - Polygon zkEVM post

Arbitrum

- Arbitrum API Quickstart
-
  - Arbitrum SDK Examples
- Arbitrum API FAQ
-
  - Arbitrum vs. Ethereum API Differences
- Arbitrum API Endpoints
-
  - eth_call - Arbitrum post
-
  - eth_estimateGas - Arbitrum post
-
  - eth_accounts - Arbitrum post
-
  - eth_blockNumber - Arbitrum post
-
  - eth_chainId - Arbitrum post
-
  - eth_gasPrice - Arbitrum post
-
  - eth_getBalance - Arbitrum post
-
  - eth_getBlockTransactionCountByHash - Arbitrum post
-
  - eth_getBlockTransactionCountByNumber - Arbitrum post
-
  - eth_getCode - Arbitrum post
-
  - eth_getFilterChanges - Arbitrum post
-
  - eth_getFilterLogs - Arbitrum post
-
  - eth_getLogs - Arbitrum post
-
  - eth_getStorageAt - Arbitrum post
-
  - eth_getTransactionByBlockHashAndIndex - Arbitrum post
-
  - eth_getTransactionCount - Arbitrum post
-
  - eth_getUncleByBlockNumberAndIndex - Arbitrum post
-
  - eth_getUncleCountByBlockHash - Arbitrum post
-
  - eth_getUncleCountByBlockNumber - Arbitrum post
-
  - eth_newBlockFilter - Arbitrum post
-
  - eth_newFilter - Arbitrum post
-
  - eth_newPendingTransactionFilter - Arbitrum post
-
  - eth_uninstallFilter - Arbitrum post

-
  - net_listening - Arbitrum post
-
  - net_version - Arbitrum post
-
  - web3_clientVersion - Arbitrum post
-
  - web3_sha3 - Arbitrum post
-
  - eth_sendRawTransaction - Arbitrum post
-
  - eth_createAccessList - Arbitrum post
-
  - eth_maxPriorityFeePerGas - Arbitrum post
-
  - eth_feeHistory - Arbitrum post
-
  - eth_getBlockByHash - Arbitrum post
-
  - eth_getBlockByNumber - Arbitrum post
-
  - eth_getTransactionByBlockNumberAndIndex - Arbitrum post
-
  - eth_getTransactionByHash - Arbitrum post
-
  - eth_getProof - Arbitrum post
-
  - eth_getTransactionReceipt - Arbitrum post
-
  - eth_getUncleByBlockHashAndIndex - Arbitrum post
-
  - eth_subscribe
-
  - eth_unsubscribe

Optimism

- Optimism API Quickstart
-
  - Optimism SDK Examples
- Optimism API FAQ
-
  - Optimism Error Codes
- Optimism API Endpoints
-
  - eth_call - Optimism post
-
  - eth_estimateGas - Optimism post
-
  - eth_accounts - Optimism post
-
  - eth_blockNumber - Optimism post
-
  - eth_chainId - Optimism post
-
  - eth_gasPrice - Optimism post
-
  - eth_getBalance - Optimism post
-
  - eth_getBlockTransactionCountByHash - Optimism post
-
  - eth_getBlockTransactionCountByNumber - Optimism post
-
  - eth_getCode - Optimism post
-
  - eth_getFilterChanges - Optimism post
-
  - eth_getFilterLogs - Optimism post
-
  - eth_getLogs - Optimism post
-
  - eth_getStorageAt - Optimism post
-
  - eth_getTransactionByBlockHashAndIndex - Optimism post
-
  - eth_getTransactionByBlockNumberAndIndex - Optimism post
-
  - eth_getTransactionByHash - Optimism post
-
  - eth_getTransactionCount - Optimism post
-
  - eth_getTransactionReceipt - Optimism post
-
  - eth_getUncleByBlockHashAndIndex - Optimism post
-
  - eth_getUncleByBlockNumberAndIndex - Optimism post
-
  - eth_getUncleCountByBlockHash - Optimism post
-
  - eth_getUncleCountByBlockNumber - Optimism post
-
  - eth_newBlockFilter - Optimism post
-
  - eth_newFilter - Optimism post
-
  - eth_newPendingTransactionFilter - Optimism post
-
  - eth_protocolVersion - Optimism post
-
  - eth_sendRawTransaction - Optimism post
-
  - eth_syncing - Optimism post
-
  - eth_uninstallFilter - Optimism post
-
  - net_listening - Optimism post
-
  - net_version - Optimism post
-

  - web3_clientVersion - Optimism post
-
  - web3_sha3 - Optimism post
-
  - eth_getBlockByHash - Optimism post
-
  - eth_getBlockByNumber - Optimism post
-
  - eth_getProof - Optimism post
-
  - eth_subscribe
-
  - eth_unsubscribe

Base

- Base API Quickstart
- Base API FAQ
- Base API Endpoints
-
  - eth_accounts - Base post
-
  - eth_blockNumber - Base post
-
  - eth_call - Base post
-
  - eth_chainId - Base post
-
  - eth_estimateGas - Base post
-
  - eth_feeHistory - Base post
-
  - eth_gasPrice - Base post
-
  - eth_getBalance - Base post
-
  - eth_getBlockByHash - Base post
-
  - eth_getBlockByNumber - Base post
-
  - eth_getBlockTransactionCountByHash - Base post
-
  - eth_getBlockTransactionCountByNumber - Base post
-
  - eth_getCode - Base post
-
  - eth_getFilterChanges - Base post
-
  - eth_getFilterLogs - Base post
-
  - eth_getLogs - Base post
-
  - eth_getProof - Base post
-
  - eth_getStorageAt - Base post
-
  - eth_getTransactionByBlockHashAndIndex - Base post
-
  - eth_getTransactionByBlockNumberAndIndex - Base post
-
  - eth_getTransactionByHash - Base post
-
  - eth_getTransactionCount - Base post
-
  - eth_getTransactionReceipt - Base post
-
  - eth_getUncleByBlockHashAndIndex - Base post
-
  - eth_getUncleByBlockNumberAndIndex - Base post
-
  - eth_getUncleCountByBlockHash - Base post
-
  - eth_getUncleCountByBlockNumber - Base post
-
  - eth_maxPriorityFeePerGas - Base post
-
  - eth_newBlockFilter - Base post
-
  - eth_newFilter - Base post
-
  - eth_newPendingTransactionFilter - Base post
-
  - eth_protocolVersion - Base post
-
  - eth_sendRawTransaction - Base post
-
  - eth_syncing - Base post
-
  - eth_uninstallFilter - Base post
-
  - net_listening - Base post
-
  - web3_sha3 - Base post

❇ Solana

- Solana API Quickstart
- Solana API FAQ
- Solana API Endpoints
-
  - getAccountInfo post
-
  - simulateTransaction post
-
  - getBalance post
-
  - getBlock post
-
  - getBlockCommitment post

What are EVM Traces?

A guide to understanding EVM traces, their types, and how to use them.

Prerequisites

Before reading about EVM traces, you should have a clear understanding of EVM , Execution Clients , Smart Contracts and Nodes . You can read about these topics in the official Ethereum documentation orAlchemy documentation . However, if you just need a refresher, you can find the one-line definitions for these topics below.

- Smart Contracts:
- Smart contracts are self-executing contracts with the terms of the agreement written into lines of code. The code and the agreements contained therein exist across a decentralized, distributed blockchain network.
- Execution Client:
- An Execution/Ethereum Client is a software program that connects to the Ethereum network to enable users to interact with the Ethereum blockchain. Some popular Ethereum clients include Geth, Parity, and Erigon.
- EVM:
- The Ethereum Virtual Machine (EVM) is a turing-complete virtual machine that allows for the execution of smart contracts on the Ethereum blockchain. The EVM is responsible for processing and executing all of the transactions that occur on the Ethereum network.
- Node:
- An Ethereum node is a computer that runs an Ethereum Client and maintains the Ethereum blockchain.

The Problem

There are two types of transactions in EVM-compatible protocols:

1. Value Transfers:
2. A value transfer just moves the native blockchain currency (Ether in the case of Ethereum) from one account to another.
3. Contract Executions:
4. A contract execution involves calling a function of the smart contract that can change the state of the contract and even call functions of other smart contracts.

The downside of the contract execution is that it is very hard to tell what the transaction actually did. When a transaction is executed, you can get a transaction receipt that contains a status code to check whether the execution succeeded or not, besides looking at EVM traces there is no way to see what data was modified, or what external contracts were invoked.

Look at the transaction flow below:

Here is theEtherscan link for the transaction defined above. This is a transaction in which a user (externally owned account) invokes a contract which in turn invokes another contract and this is how the transaction receipt for this transaction looks like:

transaction-receipt.json {to :'0xdE8BFD2b6E8F765710198fd10E7e87757B959F84' ,from :'0x80b4f0bc53F620141C16Bd209269aeC0D72B22c4' ,contractAddress :null ,gasUsed :BigNumber {_hex :'0x7f54' ,_isBigNumber :true },blockHash :'0x985ab37352c3c8765c6f7b480e07e1eadef6dd53c06fa25cf72394cb8eae34' ,transactionHash :'0x0d13800319458a403500d407226328bbedbd8a10ff3d26a18c67872ac1f94ea7' ,blockNumber :15095426 ,confirmations :768898 ,effectiveGasPrice :BigNumber {_hex :'0x0454f97690' ,_isBigNumber :true },status :1 , }

NOTE

You can get the transaction receipt for any transaction using theeth_getTransactionReceipt Alchemy API The transaction receipt has a couple of fields, let's take a look at them one by one:

- to
- : The address the transaction is directed to. (In this case, Contract A address).
- from
- : Address of the caller (In this case, the user).
- contractAddress
- : If a new contract was created as part of this transaction then this field would have the address of the newly created contract, in this case, it is
- null
- because no new contract was created.
- gasUsed
- : Gas used for the execution of the transaction in big number format.
- blockHash
- : Hash of the block in which the transaction is included.
- transactionHash
- : The hash of the transaction itself. This is the unique identifier of the transaction.
- blockNumber
- : The block number in which the transaction is included.
- confirmations
- : The number of blocks that have been mined since the block containing the given transaction was mined.
- effectiveGasPrice
- : Gas price for the transaction.
- status
- : A status with the value
- 1
- means that the transaction was successful while a value of
- 0
- means that the transaction failed.

As you can see that the transaction receipt provides some information about the transaction but it does not tell that contract A further called contract B. This can be a problem depending on your use case. If you want information about all the steps involved in the transaction including calls to other contracts then transaction receipt will not be of much use to you. So how can this problem be solved?

The Solution: EVM Traces

EVM traces provide a step-by-step record of what happened during the execution of a transaction, including which other contracts were invoked and what data was changed. They help give debugging information that can be used to troubleshoot issues with EVM-compatible smart contracts, which is useful for understanding why a contract behaved in a certain way, or for finding bugs.

When you trace a transaction you get back an array of objects also known asEVM traces for the transaction. Each step of the transaction is represented as an individual object in the array.

Example

So, the EVM traces forthis transaction which is also defined in "The Problem" section looks like this (We'll learn how to retrieve EVM traces in a later section):

evm-traces.json [{"action" : {"from" :"0x80b4f0bc53f620141c16bd209269aec0d72b22c4" ,"callType" :"call" ,"gas" :"0x7a37" ,"input" :"0x" ,"to" :"0xde8bfd2b6e8f765710198fd10e7e87757b959f84" ,"value" :"0xb2ece213edb23a" },"blockHash" :"0x985ab37352c3c8765c6f7b480e07e1eadef6dd53c06fa25cf72394cb8eae32b4" ,"blockNumber" :15095426 ,"result" : {"gasUsed" :"0x2d4c" ,"output" :"0x" },"subtraces" :1 ,"traceAddress" : [],"transactionHash" :"0x0d13800319458a403500d407226328bbedbd8a10ff3d26a18c67872ac1f94ea7" ,"transactionPosition" :420 ,"type" :"call" }, {"action" : {"from" :"0xde8bfd2b6e8f765710198fd10e7e87757b959f84" ,"callType" :"call" ,"gas" :"0x8fc" ,"input" :"0x" ,"to" :"0xaf1931c20ee0c11bea17a41bfbbad299b2763bc0" ,"value" :"0xb2ece213edb23a" },"blockHash" :"0x985ab37352c3c8765c6f7b480e07e1eadef6dd53c06fa25cf72394cb8eae32b4" ,"blockNumber" :15095426 ,"result" : {"gasUsed" :"0x0" ,"output" :"0x" },"subtraces" :0 ,"traceAddress" : [0],"transactionHash" :"0x0d13800319458a403500d407226328bbedbd8a10ff3d26a18c67872ac1f94ea7" ,"transactionPosition" :420 ,"type" :"call" }] Here is an explanation of the above EVM traces (click on the image to zoom into it):

Each step of the transaction is represented as a separateTrace and each trace has its own fields. A lot of the fields are the same as the fields of the transaction receipt, such as from , to , blockHash , blockNumber and transactionHash but there are additional fields as well.

Thetype field defines the type of the given action/trace and the structure of the given trace depends on the type of the action. Let's take a look at the types of trace actions and their structures.

Types of Trace Actions

There are many types of actions captured in transaction traces: Some common actions areCREATE ,SUICIDE andCALL . Below you will find the structure for these trace actions.

CREATE

Captured when a new smart contract is created.

Structure

- action
-

    - from
-
    - : address that created the contract
-
    - gas
-
    - : gas cost to create the contract
-
    - init
-
    - : initialization code for creating the contract
-
    - value
-
    - : value sent to contract
- blockHash
- : block hash the transaction was included in
- blockNumber
- : block number the transaction was included in
- result
-
    - address
-
    - : address for contract created
-
    - code
-
    - : code for contract created
-
    - gasUsed
-
    - : gas used in contract creation
- subtraces
- : number of child traces of the given trace
- traceAddress
- : index for the given trace in the trace tree
- transactionHash
- : hash for the transaction
- transactionPosition
- : position (or index) of transaction in the block
- type
- : type of action, in this case,
- CREATE

Example:

create-evm-trace.json {"action" : {"from" :"0x6090a6e47849629b7245dfa1ca21d94cd15878ef" ,"gas" :"0x6a7f1" ,"init" :"0x606060405260405160208061051683398101604052515b60028054600160a060020a03808416600160a060020a0319928316179092556000805433909316929091169190911790554260019081556 ,"value" :"0xe4b4b8af6a70000" },"blockHash" :"0x6d00f7707938cca36b0730d8f7f090543242002b6fa0fe94bf85b9ab02e6bed6" ,"blockNumber" :4000036 ,"result" : {"address" :"0xfc9779d9a0f2715435a3e8ebf780322145d7546e" ,"code" :"0x606060405236156100885763ffffffff60e060020a600035041663305b34410811461008a5780630b5ab3d5146100ac57806313af4035146100be5780632b20e397146100dc5780633fa4f24514610108578 ,"gasUsed" :"0x52ce0" },"subtraces" :0 ,"traceAddress" : [0],"transactionHash" :"0xc9601ea5ca42e57c3ef1d770ab0b278d6aadf2511a4feb879cba573854443423" ,"transactionPosition" :70 ,"type" :"create" } Here is thelink for the transaction on Etherscan whose example EVM trace is given above.

SUICIDE

Captured when a smart contract is destroyed, which transfers the contract's current balance to a specified address and clear the contract's data, freeing up memory on-chain. The freed space on-chain is processed as a refund towards the total gas cost for completing the transaction.

Structure

- action
-
    - address
-
    - : address of contract to destroy
-
    - refundAddress
-
    - : address to send the remainder of the contract
-
    - balance
-
    - to
-
    - balance
-
    - : remaining balance in the contract
- blockHash
- : block hash the transaction was included in
- blockNumber
- : block number the transaction was included in
- result:
- null
- for
- SUICIDE
- actions
- subtraces
- : number of child traces of the given trace
- traceAddress
- : index for the given trace in the trace tree
- transactionHash
- : hash for the transaction
- transactionPosition
- : position (or index) of transaction in the block
- type
- : type of the action, in this case,
- SUICIDE

Example:

suicide-evm-trace.json {"action" : {"address" :"0x87051f6ba0562fdb0485763562bf34cb2ad705b1" ,"refundAddress" :"0x000000000000000000000000000000000000dead" ,"balance" :"0x0" },"blockHash" :"0x6d00f7707938cca36b0730d8f7f090543242002b6fa0fe94bf85b9ab02e6bed6" ,"blockNumber" :4000036 ,"result" :null ,"subtraces" :0 ,"traceAddress" : [1 ,2 ,2],"transactionHash" :"0xbc15addb97490a168dc1d099ab8537caf2e4ff7d1deeff6d685d2d594a750037" ,"transactionPosition" :45 ,"type" :"suicide" }, Here is thelink for the transaction on Etherscan whose example EVM trace is given above.

CALL

Used for transferring ETH betweenexternally owned accounts (EOAs) or to call a smart contract function.

Structure

- action
-
    - from
-
    - : address of the sender
-
    - callType
-
    - : type of
-
    - CALL
-
    - , can be any of the following:* call
-
    -
        - delegatecall
-
    -
        - callcode
-
    -
        - staticcall
-
    - gas
-
    - : gas included in the transaction
-
    - input
-
    - : the specific function to call on the contract with parameters specified, encoded. For transfers to an EOA,
-
    - input
-
    - will be
-
    - 0x
-
    - to
-
    - : address the transaction is directed.
-
    - value
-
    - : the amount of value to be transferred
- blockHash
- : block hash the transaction was included in
- blockNumber
- : block number the transaction was included in
- result
-
    - gasUsed
-
    - : gas used to execute the transaction
-
    - output
-
    - : the result of the smart contract function call, encoded. For transfers to an EOA or smart contract, the
-
    - output
-
    - will be
-
    - 0x
-
    - .
- subtraces
- : number of child traces of the given trace
- traceAddress
- : index for a given trace in the trace tree
- transactionHash
- : hash for the transaction
- transactionPosition
- : position (or index) of transaction in the block
- type
- : type of the action, in this case,
- CALL

Example:

call-evm-trace.json {"action" : {"from" :"0xbc9f06dd67578b0b8b4d87fda9acde453bc4c067" ,"callType" :"call" ,"gas" :"0x97478" ,"input" :"0xfebefd6100000000000000000000000000000000000000000000000000000000000000040cc849afc28894f79411f12309e75c71ded27d1666b75a2423633c204e671cb1e000000000000000000000000 ,"to" :"0x6090a6e47849629b7245dfa1ca21d94cd15878ef" ,"value" :"0x2386f26fc10000" },"blockHash" :"0x6d00f7707938cca36b0730d8f7f090543242002b6fa0fe94bf85b9ab02e6bed6" ,"blockNumber" :4000036 ,"result" : {"gasUsed" :"0x7ad71" ,"output" :"0x" },"subtraces" :4 ,"traceAddress" : [],"transactionHash" :"0x552b31a3a9c92577d65db62cf9f729e81571e10cad90e356423adcfa2caebacc" ,"transactionPosition" :71 ,"type" :"call" } Here is thelink for the transaction on Etherscan whose example EVM trace is given above.

How to read

traceAddress ?

Traces are structured in a tree format. This helps in better understanding the flow of the transaction. The traceAddress field represents the position of the given trace in the tree. An empty array represents the root of the tree (the first trace). Furthermore, traces which are captured due to the first trace have their traceAddress in [0], [1], [2] etc. format.

Here is a diagram of traceAddress results to help understand how to read this position:

Applications of EVM Traces

There are many use cases for EVM traces some of them are listed below:

Transaction Tracers

Etherscan and other transaction tracers liketxs.fyi help us better understand the flow of a transaction. They extract the EVM traces for a transaction and display them in a way that’s readable by us. For example here is the result for a USDT transfer transaction ontxs.fyi .

As you can see it’s clear from the execution trace that the caller called the transfer function of the TetherToken contract and the contract transferred 285 USDT from the caller to the target address.

Debugging Transactions

When a transaction fails, you can find the reason for the failure of the transaction using EVM traces. For example, in the trace image below you can see that the transaction failed due to an "Out of gas" exception, which means that there was not enough gas to complete the transaction.

Here is thelink to the above-defined transaction on Etherscan.

Contract Performance Analysis

Transaction traces can be used to analyze the performance of smart contracts by looking at the number of actions it takes for each transaction to be processed. This information can be used to identify bottlenecks and optimize the contract for better performance.

How to retrieve EVM traces?

There are several ways to retrieve EVM traces of a transaction.

1. Using Alchemy APIs:
2. Alchemy manages trace-enabled nodes and offers API endpoints to collect transaction traces. This is the simplest way of retrieving EVM traces since running your own node requires a lot of resources and maintenance. You can either use theTrace API
3. endpoints or theDebug API
4. endpoints to get the transaction traces. For getting the transaction traces using a transaction hash, you can use thetrace_transaction
5. method.
6. Replaying the transaction in a full/archive node:
7. Ethereum clients have methods that allow them to re-run transactions that have been executed previously. They collect traces from these transactions in order to get results. Even though it takes time to retrieve the results, nodes are not required to store the traces as long as they have enough information to run the transaction.
8. Running an archive node with traces enabled:
9. Ethereum clients support running nodes with traces enabled. This allows nodes to store traces so that they can be retrieved quickly without having to re-execute the transaction. However, this comes at the expense of higher costs and slower node performance.

Conclusion

In conclusion, EVM transaction traces are a valuable tool for debugging smart contracts. They provide a step-by-step record of the execution of a contract and can be used to identify errors and optimize code.

Updated 5 months ago