Cloudflare ZT VPC
Private VPS | Ryzen 9 5950X 16c/32t | 128GB | 1 Gbit | 2x U.2 NVMe 3,84 TB | [128GB] 4x RAM ECC
Public Network
geth p2p connections
PEERS=400
FULL NODE [BNB CHAIN]
RPC Service [PORT:8545]
--http.addr "127.0.0.1" --http.port 8545
WebSocket Service [PORT:8546]
--ws.addr "127.0.0.1" --ws.port 8546
Metric Service [PORT: 8086]
--metrics.influxdb
IPC Service [geth.ipc]
/home/fullnode/chaindata/snapshot/
geth.pbss/geth/geth.ipc
Cloudflared [Zero Trust Network]
HTTP/RPC Service [https://bsc-rpc.prxo.io]
WebSocket Service [wss://bsc-wss.prxo.io]
Grafana Metrics [metrics.prxo.io]
Docker [containers]
Grafana Metrics
http://localhost:3000
InfluxDB & Prometheus
http://localhost:8086
Redis Stack Server
redis://:<password>@localhost:6379
Microservice 01 - [Rust] Transaction Scanner/Filter → Redis
main()
Filters every new block (and pending transactions) in realtime. Stores all transaction objects that match the specified contract address. Pending transactions are stored first, once confirmed, the pending transaction will be overwritten with its confirmed TxObject.
load_config(), start process_headers(), start process_pending(), provide conection
process_headers() - [confirmed tx]
Params
-target_address: H160,
-redis::Connection,
-ipc_path: String,
Operations
+ log_transaction(redis, tx.clone())
process_pending() - [pending tx]
Params
-target_address: H160,
-redis::Connection,
-ipc_path: String,
Operations
+ log_transaction(redis, tx.clone())
log_transaction() -> Redis
Inputs
con (redis::Connection) | tx (tx object to log) | type_id (string: confirmed/pending)
Operations
+ json_set(key, path, tx_object)
Main Service - [Bun Runtime]+[TypeScript] - Trade Controller
index.ts [main]
Inputs
-redis connection string
Operations
+ startPolling() - stores assetIndex prices
+ Redis Keyspace Channel - Event Subscription
+ Redis Controller (input)
+ decodeTx(tx.input)
+ handleTx(obj)
Redis Connection Controller
index.ts [decodeTx()]
Decodes all transaction input values.
Inputs
-abi.json
-methodDirectory.json
Return - {object}
+ methodSignature,
+ decodedInput,
+ humanReadableDecodedInput
index.ts [redis controller]
Log New Transaction -> Redis JSON
index.ts [startPolling()]
Stores realtime asset price data in Redis
https://www.apollox.finance/fapi/v1/assetIndex
Operations
+ startPolling() -> fetchAndStoreData()
Fetch data at set intervals, JSON.set -> Redis
index.ts [redis subscription]
Subscribes to new json keys that have been stored in Redis by the rust TxScanner
Operations
+ handleTx(tx.input)
index.ts [handleTx()]
Decode Tx, Store Tx, Start Trade
Params
+ handleTx(txObject)
Operations
+ decodeTx = parseInputData(tx.input)
Start Trade IF MethodSignature contains 'openMarketTrade'
Params
+ decodeTx(returnedTxObject)
Operations
+ IF openMarketTrade (startTrade(redisData))
+ Store new transaction in Redis as a copy trade
components/trade/startTrade.ts [startTrade(tx)]
Check if the tx.from address matches any address found in traders.json
Params: (tx: Transaction)
Operations
+ checkForAddressMatch(tx, traders:Trader[])
IF Trade is a match:
Operations
+ handleMatch(match:Trader, tx:Transaction)
components/trade/startTrade.ts [handleMatch(,)]
Checks that trade matches BTC ETH or BNB, then calls processTransaction with the parameters for the detected pair
Params: (trader: Trader, tx: Transaction)
Operations
+ processTransaction(tx,845,'USDT',ADDRESS,TP,SL)
[CONTROLLER]
components/trade/startTransaction.ts
processTransaction()
Main function to process and submit the transaction
Params:
-transaction: Transaction,
-lev: number,
-token: string,
-baseToken: string,
-takeProfitPercent: number,
-stopLossPercent: number
Operations
+ myAmounts(tokenInPrice,lev,token)
+ calculateQuantity(totalValue,price)
+ debugTransaction()
+ getTradeHashFromReceipt(receipt)
+ getEntryPrice(tradeHash)
+ calculateTakeProfitPrice(params.......)
+ updateTradeTp(tradeHash,takeProfitPrice)
+ saveToRedis(redisData, path)
Redis Connection Controller
myAmounts(tokenInPrice,lev,token)
Converts numbers, calculates main trade paramaters
Operations
+ convertToWei(value:BN, decimals:number)
+ handleAmountIn()
returns { myAmountInWei, amountInUSD }
calculateQuantity(totalValue,price)
Calculates the quantity based on the total value and price.
Params: (totalValue:string|number, price:string)
Returns qtyNormalized (ABI compliant)
processTransaction(params...)
Submits the transaction to an opBNB RPC endpoint
Params: (same as parent)
Returns receipt
getTradeHashFromReceipt(receipt)
Retrieves the trade hash from a transaction receipt by decoding the logs.
returns tradeHash:string
updateTradeTp(tradeHash,takeProfitPrice)
Updates the take profit price of the trade
Params: tradeHash, takeProfitPrice
Returns: receipt
@utils/convertToWei.ts (value,decimals)
Converts values to wei
Params: (value:BN, decimals:number)
@trade/startTransction.ts [handleAmountIn()]
Calculates percentage of wallet balance to spend on the trade
Returns a decimal number in USDT
@trade/getPosition.ts [pollForEntryPrice()]
Polls getPositionByHashV2 via RPC until the transaction is confirmed and receives a response
returns 'position' object
getEntryPrice(tradeHash)
Get market entry price for confirmed position
Operations
+ pollForEntryPrice(tradeHash:string)
calculateTakeProfitPrice(params.......)
Calculates the take profit price based on the initial margin, leverage, BTC buy price, and take profit percentage.
Params:
-initialMargin: number,
-leverage: number,
-btcBuyPrice: number,
-takeProfitPercentage: number,
-long: boolean
returns 'takeProfitPrice:number'
External opBNB RPC Endpoint
from .env