Ethereum Bytecode Database Microservice

Database with shared contract information

Blockscout supports multiple chains, and each chain has its own database that stores verified smart-contracts. Because these databases are independent of one another, verified contracts and their method identifiers typically cannot be shared between chains.

For example, if a contract is verified on Ethereum Mainnet, it cannot be accessed on Gnosis Chain even if the creation transaction input is the same.

?

To address this issue, we have implemented a database for verified contract information which can be shared across all supported chains. This database and it's associated functionality has been created as a Blockscout microservice called theEthereum Bytecode Database (Blockscout EBD).

Initial Planning

Initial plans were to create a distinct service with a database responsible for maintaining verification results. This database could then be shared among all supported chains. The service would track the contract address and the chain where the contract was deployed. The source files would also be stored so they could be extracted for verified contracts. The service would also find and match unverified contracts to their verified counterparts if they already existed in the database.

?

Restructuring the Search Process

It is possible to search for similar bytecodes if the bytecodes (or their hashes) are stored in the database along with verification results. The list of already verified bytecodes can then be checked when searching for the source codes of unverified contracts.

However, with this type of search thechainId andcontractAddress arguments are typically required, as source codes for already verified contracts are stored based on the chain and address where these contracts are deployed. This is how most source code databases are designed (e.g., Sourcify, Etherscan, Blockscout).

With this new service, our main goal was not just to return source codes for verified contracts (since that data is already stored and processed internally within Blockscout's main storage), but also to search for source codes for unverified contracts. To do this, we realized we could eliminatechainId andcontractAddress and gain some advantages.

Solution - Ethereum Bytecode Database (Blockscout EBD)

A solution can be found by mapping source codes directly to bytecodes. In this case,chainId andcontractAddress are not needed, only thebytecode - sourceCode correspondence if required. This is similar to what4bytes orsigEth does. However, instead of looking for the function signature based on its identifier, we look for the contracts' source code based on its bytecode.

- Current Approaches:
- Chain → Contract Address → Sources
- Blockscout Ethereum Bytecode Database:
- Bytecode → Sources
-

Basics

In the following diagram for the new service, notice that there is no notion ofchainId andcontractAddress —only raw bytecodes are important.

Verification

?

Bytecode Source Search

The microservice does not care where the contract is deployed (which chain) or even whether a contract with the corresponding bytecode is deployed at all.

?

Similar Contracts Search Enhancement

In Blockscout, two contracts share the same source code only if their bytecodes are entirely identical. However, not every part of the bytecode is functional (seeSourcify partial vs. full match). There may also be a metadata hash, usually located at the end of the creation input or deployed bytecode. The source code can also produce different metadata hashes for inconsequential changes, such as when a new space is appended to the file or the file is renamed. This results in bytecodes with the same EVM-executed portion, but they may not be identified by Blockscout's current similar contracts search algorithm.

As described insmart-contract verification , our verification service allows us to split the bytecode into 2 parts,Main andMetadata . The Ethereum Bytecode Database takes advantage of this and searches for similar bytecodes based only on theMain (functionally significant) parts.

Adding Extractors

Removing the notion of chains and contract addresses is powerful in and of itself. However, the addition of extractors provides the opportunity for an extremely robust database.

?

Extractors can be implemented as separate extensions which can index different explorers and different chains, then automatically submit newly verified bytecodes into the Ethereum Bytecode Database. It can also import contracts verified long ago.

This has the potential rapidly increase the dataset,allowing the service to collect sources for almost all bytecodes that have been publicly verified . Every time a new contract is verified on a monitored explorer, the extractor can upload the source code to the Ethereum Bytecode Database.This is in the research phase.

In addition, anyone can start the service from scratch and eventually obtain their own repository of verified contracts by running extractors they are interested in. Those who would like to host their own instance can obtain a populated database which can also be updated from other chains in the ecosystem.

Bytecode DB in Action

In this video we show a simple example where a contract is deployed and verified on Optimism Goerli. The same contract is then deployed to the Base Goerli instance, and the contract is automatically verified using the Blockscout Bytecode Database.

?

Conclusion

Implemented as a separate service,Ethereum Bytecode Database may become a unique source of verified contracts working with many chains. Furthermore, the addition of extractors allows verified contracts to be combined from an unlimited number of explorers, and gives users the ability to run the service locally using independently-obtained data.

Extractors implemented within a Blockscout private setting also allow for the creation of a unique and useful ecosystem-wide database. This will likely create a data source that will be attractive to different explorers and other data aggregators. We are currently looking into the best way to provide 3rd party access to this dataset, possibly as a paid database or verifier service in the future.

Implementation Details

API Integration

The swagger definition -https://app.swaggerhub.com/apis/rimrakhimov/EthereumBytecodeDatabase/v2 The service has the same public API as thesmart-contract-verifier service. If a chain wants to start saving verified bytecodes into a database, the only required change is to pointRUST_VERIFICATION_SERVICE_URL to theeth-bytecode-db service instead of the verifier. The service will proxy all requests to the underlying verifier internally.

Completed Processes

1. TheEth-bytecode-db
2. service is implemented with both contract verification proxy and database search functionality (https://github.com/blockscout/blockscout-rs/tree/main/eth-bytecode-db
3.).
4. All hosted blockscout instances are usingeth-bytecode-db
5. instead of the original smart-contract-verifier; all newly verified bytecodes and corresponding sources are now stored directly in the database.
6. Search functionality is integrated. (https://github.com/blockscout/blockscout/pull/7187
7.).
8. DB is available on production instances including Ethereum Mainnet, Goerli, Gnosis, Optimism and more.

9. Extractor has been implemented for the "smart-contract-fiesta
10. " dataset.
11.

Current production database contains >130k source codes, where ~100k have unique main parts.

Future Plans

1. Implement a Blockscout extractor and import all previously verified contracts from Blockscout instances into the runningeth-bytecode-db
2. database (should expand the current database up to ~100-150k contracts).
3. Estimate and implement prioritization techniques, so that the service returns potential contracts that are the most probable for the given bytecode first.
4. Create a separate page similar tohttps://www.4byte.directory/
5. which allows users to search for the source code for a given bytecode directly, and submit new contracts into the database (to better understand how the service works and expand our database).
6. Support verification of contracts verified via metadata. Write a Sourcify extractor to import these contracts into our database (should expand the database up to ~400-500k contracts).
7.

Last updated8 months ago