We’re excited to announce rbuilder now supports running in-process with reth!

Check out the new [reth-rbuilder

crate](https://github.com/flashbots/rbuilder/tree/develop/crates/reth-rbuilder) for an example of how to run it, and for the curious out there the code deep-dive screencast for a technical deep dive.

1. Introduction: From Out-of-Process to In-Process for rbuilder

rbuilder has traditionally operated out-of-process, meaning it runs as a separate process that communicates with the main reth client but disconnected from direct in-memory state access. While this approach works, transitioning rbuilder to an in-process mode

would streamline access to state, reduce inter-process communication overhead, optimize performance, and allow more tightly integrated block building strategies.

In this post, we’ll walk through how rbuilder operates out-of-process, the vision for supporting in-process, and the technical hurdles we faced in this transition. Finally, we’ll dive into the code changes that made this possible and demonstrate rbuilder in action in-process.

2. How rbuilder Currently Operates Out-of-Process

In its out-of-process setup, rbuilder’s entry point lies within the LiveBuilder, where configuration is loaded, services are spun up, and the primary builder component is initialized:

// Entry point for rbuilder CLI async fn run() { ... let provider_reopener = config.base_config().create_provider_factory()?; let builder = config.new_builder(provider_reopener, cancel.clone()).await?; ... builder.run().await?; ... }

The ProviderFactoryReopener

The ProviderFactoryReopener

is a core piece of the out-of-process rbuilder that manages state access by wrapping a ProviderFactory

(provides disk DB access). ProviderFactory

is essential for connecting to the database or reading static files, but it also has limitations—particularly, only one database connection is allowed per process, restricting our ability to initiate another provider instance in-process with reth.

The ProviderFactoryReopener

is used as a workaround for Block hash data inconsistency when reading data from a different process that leads to incorrect EVM execution. · Issue #7836 · paradigmxyz/reth · GitHub.

3. Our Vision for In-Process rbuilder

Switching rbuilder to in-process mode would minimize communication overhead, provide faster, more efficient state interactions, and enable tighter block building integration with reth.

Initial Approach

Initially, we hoped to reference the ProviderFactory

from reth and use it to build a ProviderFactoryReopener

.

However, this proved to not be possible due to the reth node not exposing public api access to the ProviderFactory

. It would cause issues with future reth versions, which do not flush block state to disk each block instead keeping recent blocks in-memory only.

Adapting to Reth Provider Traits

Instead, reth’s node exposes the database and state access through traits. We needed to implement traits like StateProviderFactory

, HeaderProviderTrait

, and DatabaseProviderFactory

on our ProviderFactoryReopener

and then refactor our code to use the traits instead of any concrete struct to support both out-of-process ProviderFactoryReopener

and in-process reth node provider.

Refactoring for Flexibility

The hardcoded ProviderFactoryReopener

dependencies required extensive refactoring to allow rbuilder to work with generics for its database (DB) and provider (P) components. This refactoring enabled compatibility with both out-of-process and in-process modes without impacting existing functionality.

// Example: Refactored with Generics for Provider and Database implLiveBuilder where P: ProviderTraits, DB: DatabaseTraits, { ... }

The Minimal reth-rbuilder

Crate

We created a new crate, reth-rbuilder

, which demonstrates starting an in-process rbuilder. In this setup, rbuilder launches with the node's provider context, enabling seamless state access and tight integration with reth internals.

5. In-Process rbuilder Demo

With these changes in place, running rbuilder in-process alongside reth is straightforward. Here's a demonstration of the setup:

1. Initialize Reth Node

with rbuilder configuration.

- Configure Reth with the rbuilder_config

option for in-process rbuilder integration.

- Adjust rbuilder's ports if conflicting with other local nodes.

cargo run --bin reth-rbuilder -- node --rbuilder.config config-playground.toml --discovery.port 30304 --port 30304 --authrpc.port 8552

1. Configure Reth with the rbuilder_config

option for in-process rbuilder integration.

1. Adjust rbuilder's ports if conflicting with other local nodes.
2. Run rbuilder In-Process

using the node provider.

- Watch as rbuilder initializes and accesses the MEV-Boost data, transaction pool, and RPC services in-process.
- Watch as rbuilder initializes and accesses the MEV-Boost data, transaction pool, and RPC services in-process.