Flash Loan Basics (Balancer)

To recap, flash loans allow us to borrow vast amounts of tokens, which we can use to pay for trades. If we make a profit from executing an arbitrage, we pay the loan back and keep what's left over. If the arbitrage fails or does not produce enough profit to repay the loan, the transaction reverts and Flashbots prevents it from landing onchain, so we don't pay any fees. The only fees we pay are gas fees for successful arbitrage trades.

Flash loans work because oftransaction atomiticy . Ethereum implements this notion of atomicity over a single transaction: either all steps of the transaction succeed, or the entire transaction fails. So we say "the transaction reverts." The obvious example is a flash loan: if you borrow tokens with a flash loan, and then exit the transaction without paying it back, that transaction would be forced to revert. This is because the flash loan code itself is designed to revert if the borrower doesn't send the borrowed tokens back.

We'll implement flash loans using Balancer because they don't charge any loan fees, and have plenty of capital with which we can perform our arbitrage trades. Typically, flash loans (including Balancer) work by using callbacks. Callbacks are just functions thatother smart contracts call onyour smart contract. A callback in solidity is implemented by declaring a function in your smart contract with the exact function signature (name, arguments, return type) of the callback. The function must also beexternal (we'll see why soon), but we won't make itpublic since we don't plan to call it ourselves.

In the case of a flash loan on Balancer, the callback we have to create is thereceiveFlashLoan function. When we want to get a flash loan, we'll callvault.flashLoan , wherevault is the Balancer Vault smart contract, which holds the capital used for flash loans. In turn, the vault contract will call thereceiveFlashLoan callback on our smart contract (this is why the function isexternal). In this function, the contract will have received the funds from the flash loan, so that's where we'll performs an arbitrage trade and pay back the loan. That's all there is to it!

Adding Flash Loans to Your Contract

Flash loan capabilities can be added to any smart contract by simply implementing thereceiveFlashLoan function. The only requirement for this function is that it pays back the flash loan, but we'll also call our arbitrage function here, since we only need flash loans for executing arbitrages. Add this function to your existing smart contract, and then add a function that callsvault.flashLoan . Here's an example we borrowed from theBalancer docs :

function

makeFlashLoan (IERC20 [ ]

memory tokens , uint256 [ ]

memory amounts , bytes

memory userData)

external

{ vault . flashLoan (this , tokens , amounts , userData) ; } We'll callmakeFlashLoan to get a flash loan, which will in turn trigger thereceiveFlashLoan function, which will execute the arbitrage trade and pay the loan back.

Here's a finished smart contract fromsimple-blind-arbitrage for reference:

BlindBackrunFlashLoan.sol // Loading https://raw.githubusercontent.com/flashbots/simple-blind-arbitrage/main/src/BlindBackrunFlashLoan.sol ... Now that our contract is ready, we need to watch for new pending transactions and send arbitrage bundles to Flashbots when we find them. See the next page for instructions on writing a bot to do this for you. Edit this page Last updatedonJan 30, 2024 Previous Breakdown: simple-blind-arbitrage Next Automated Arbitrage Bot