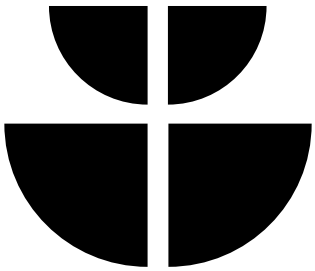
Basin: A Composable EVM-Native Decentralized Exchange Protocol



Brendan Sanderson
brendan@manifestcrypto.org

Ben Weintraub
ben@manifestcrypto.org

Brean
brean.beanstalk@protonmail.com

Silo Chad
silochad@protonmail.com

Beanstalk Farms
beanstalkfarms@protonmail.com

basin.exchange

Published: August 23, 2023
Modified: August 23, 2023
Whitepaper Version: 1.0.0
Code Version: 1.0.0[1]

"Composability is to software as compounding interest is to finance."

- Chris Dixon, October 23, 2021[2]

Abstract

Exchanging is a core piece of economic activity. However, current decentralized exchange (DEX) architectures are not composable, such that adding an exchange function or oracle to them is difficult. The nature of open source software is that each problem should only need to be solved once in a given execution environment. Non-composable DEX architectures prevent this. We propose an EVM-native DEX architecture that (1) allows for composing arbitrary exchange functions, oracles and exchange implementations for ERC-20 Standard[3] tokens and (2) includes a registry to verify exchange implementations.

1 github.com/BeanstalkFarms/Basin
2 twitter.com/cdixon/status/1451703067213066244
3 eips.ethereum.org/EIPS/eip-20

Table of Contents

1 Introduction

Despite the outsized importance of DEX protocols to the permissionless economy, innovation of them has been slow. One major problem with current decentralized exchange architectures is a lack of composability, such that to add, remove or replace a component of a decentralized exchange requires writing an entire exchange protocol. This is highly inefficient.

The primary components of a decentralized exchange protocol are (1) exchange functions, which specify the conditions under which a given asset can be exchanged for another, (2) oracles, which save data related to the exchange relevant to other protocols that need to permissionlessly and atomically query the exchange's state and (3) exchange implementations that facilitate use (*i.e.*, swapping, adding and removing liquidity) of the exchange.

Utility of decentralized exchanges has been limited in part because current DEX architectures mandate market makers provide liquidity using a finite set of exchange functions that are provided by the DEX. This limits the flexibility and capital efficiency of market makers' liquidity. Lack of flexibility and capital efficiency of liquidity make profitably market making on DEXs excessively difficult. Profitable market making is essential for well functioning liquid markets. Excessive friction to customizing exchange functions to provide liquidity on decentralized exchanges makes it difficult for decentralized permissionless trading environments to compete with centralized permissioned ones in attracting market makers. A shortage of market makers leads to a shortage of liquidity; a shortage of liquidity leads to a lack of utility for takers.

Saving oracle data on a decentralized network is expensive. The only reason to save DEX data in a network-native oracle is so that it can be queried atomically and permissionlessly by other network-native protocols. Current DEX architectures mandate using a particular oracle, independent of whether the data saved is being used by other network-native protocols. Because the cost to saving oracle data is typically passed on to takers of the DEX, this policy imposes an often unnecessary cost on all takers.

In a post-Merge[4] environment, the risk of oracle manipulation due to multi-block maximum extractable value (MEV) attacks has risen dramatically.[5] This has made using existing network-native oracle solutions risky and impractical for protocols that require manipulation resistant oracles. Upgrading the oracles in existing DEXs to be multi-block MEV resistant is impossible due to the lack of composability between exchange functions and oracles. As a result, protocols are forced to use non-network-native oracle solutions that require additional trust assumptions beyond the integrity of the network itself,[6] while takers are still paying to update useless oracles.

Basin is an open source permissionless decentralized exchange architecture that allows for the composition of arbitrary exchange functions, network-native oracles and exchange implementations into a single liquidity pool. A registry of pool implementations allows for users to verify the accuracy of a pool's use of its exchange function, oracles and exchange implementation. Anyone can deploy new exchange functions, network-native oracles, exchange implementations and registries. Similarly, anyone can deploy a new pool through an existing registry with an exchange function, any (or no) network-native oracles and an exchange implementation included in the registry. In practice, Basin lowers the friction for market makers to deploy liquidity with custom exchange functions and allows their liquidity to be used by other network-native protocols without additional trust assumptions.

4 ethereum.org/en/roadmap/merge/
5 alrevuelta.github.io/posts/ethereum-mev-multiblock
6 liquity.org/blog/price-oracles-in-liquity

2 Previous Work

Basin is the next step in the evolution of EVM-native DEXs.

A robust, trustless computer network that supports composability and fungible token standards (*e.g.*, Ethereum) is necessary to host a DEX. Basin supports the exchange of ERC-20 tokens.

Uniswap[7] pioneered use of the constant product exchange function that has been immensely popular for market making long-tail assets despite its capital inefficiency. Uniswap has also innovated on manipulation resistant network-native oracles by moving from a simple moving average (SMA) of an arithmetic mean,[8] which weights outliers equally to all other data points, to an SMA of a geometric mean,[9] which weights certain outliers significantly less.

Curve[10] implemented a variation on the constant product function that allowed for a single parameter (*i.e.*, the A parameter) to be set at pool deployment and adjusted by protocol governance. However, governance is mandatory and omnibus, such that every pool is governed by the same body, independent of who is providing liquidity to it.

Solidly[11] allowed for the use of more than one exchange function through a single DEX.

3 Basin

Well designed open source protocols enable anyone to (1) compose existing components together, (2) develop new components when existing ones fail to meet the needs of users or are cost-inefficient and (3) verify the proper use of existing components in a simple fashion. Basin allows for anyone to compose new and existing (1) *Well Functions* (*i.e.* exchange functions), (2) *Pumps* (*i.e.*, network-native oracles) and (3) *Well Implementations* (*i.e.*, exchange implementations) to create a *Well* (*i.e.*, a customized liquidity pool). *Aquifers* (*i.e.*, *Well* registries) store a mapping from *Well* addresses to *Well Implementations* to enable verification of a *Well Implementation* given a *Well* address. *Aqueducts* (*i.e.*, *Well* whitelists) store a list of approved *Well Functions*, *Pumps*, *Well Implementations* and *Aquifers* to create a single point of trust for users.

A *Well* allows for the provisioning of liquidity into a single on-chain position that follows arbitrary rules and is represented by an ERC-20 token (*i.e.*, a *Well* LP Token). Each *Well* is defined by (1) a list of tokens, which contains the set of ERC-20 tokens that the *Well* supports (*i.e.*, that can be exchanged through, added to, and removed from the *Well*), (2) a *Well Function* and its associated data, which defines an invariant relationship between the *Well's Reserves* (*i.e.*, the balances of tokens in (1) at the time of the last supported *Well* operation) and the supply of *Well* LP tokens, (3) *Pumps* and their associated data, which implement network-native oracles that are updated each time associated *Well's Reserves* may change, (4) a *Well Implementation*, which contains all logic for actions supported by the *Well*, (5) optional arbitrary additional data used by the *Well* and (6) the *Aquifer* that deployed the *Well*.

A *Well's* state consists of its *Reserve* balances and *Well* LP token ownership. *Well* LP tokens are ERC-20 tokens representing pro-rata ownership of the *Well's Reserves* and implement ERC-2612.[12] *Well Functions* and *Pumps* can be independently chosen to be stateful or stateless, while *Well Implementations* are stateful. Including *Pumps* in a *Well* is optional.

7 uniswap.org
8 uniswap.org/whitepaper.pdf
9 uniswap.org/whitepaper-v3.pdf
10 curve.fi
11 solidly.exchange/
12 eips.ethereum.org/EIPS/eip-2612

A *Well's* (1) address and its associated data, (2) list of tokens, (3) *Well Function* and its associated data, (4) *Pumps* and their associated data and (5) deploying *Aquifer*, can be stored as immutable data in contract storage or as appended bytecode. The associated data of the *Well* address, *Well Function* and *Pumps* are optional and can be arbitrary. Requirements for and use of the associated data of the *Well* address are specified by the *Aquifer*; requirements for and use of the associated data of the *Well Function* and *Pumps* are specified by the *Well Implementation*.

3.1 Well Function

A *Well Function* defines an invariant relationship between a *Well's Reserves* and the supply of *Well* LP tokens. Basin supports *Well Functions* that contain arbitrary logic. However, *Well Functions* must be deterministic to be used alongside *Pumps*. In practice, the handling of arbitrary logic in a *Well Function* allows for arbitrary order creation (i.e., conditional orders that take network state data as inputs).

3.2 Pump

Pumps are a generalized framework for network-native oracles. A *Pump* is updated upon each interaction with a *Well* that uses it. A *Pump* can be shared by multiple *Wells* if it stores a mapping of multiple *Well* addresses. Each *Pump* can determine its own method of recording a *Well's Reserves*.

When there's an interaction with a *Well*, the *Well* sends two pieces of information to its *Pumps*: (1) a list of *Reserve* amounts and (2) additional metadata as a bytestring. Each *Pump* is responsible for deciding how to weight the new *Reserve* values relative to the values it had previously and processing the bytestring.

Because storing data on-chain is expensive, only absolutely necessary data should be recorded. Each *Well* can be deployed with an arbitrary set of *Pumps*, such that only and exactly the data necessary to satisfy composability between the liquidity in the *Well* and arbitrary other protocols desired by the liquidity provider is recorded on-chain. *Pumps* create a minimalist, composable way for liquidity providers to maximize use of their liquidity in other protocols while minimizing gas costs associated with trading against it.

3.3 Well Implementation

Well Implementations contain all logic to support swapping against, adding liquidity to and removing liquidity from, a *Well* in arbitrary proportions. A *Well Implementation* may require a *Well* to have associated data which the implementation uses to properly interface with its *Well Function* (*e.g.*, an A parameter for a Curve style stableswap invariant, a trading fee, etc.) and *Pumps* (*e.g.*, the *Pump* look-back period). *Well Implementations* can specify a whitelist (or blacklist) of acceptable (or unacceptable) counterparties and/or liquidity providers.

3.4 Aquifer

Aquifer is an instance of a *Well* factory (i.e., a permissionless *Well* deployer and registry). *Aquifer* deploys *Wells* by cloning a pre-deployed *Well* and stores a mapping of the new *Well* address to the *Well Implementation* address.

Aquifer creates the first layer of trust for Basin users by providing a way to verify a *Well's Implementation*, given its address. However, users must still verify the *Well's Well Function*, *Pumps* and *Well Implementation*.

3.5 Aqueduct

Aqueduct is a whitelist of *Well Functions*, *Pumps*, *Well Implementations* and *Aquifers*. *Aqueducts* create a single point of trust for users of Basin components. As long as all the components of a *Well* are included in a trusted *Aqueduct*, the *Well* can be trusted. In practice, protocols can deploy an *Aqueduct* that specifies all the Basin components liquidity providers can (or must) use in order to receive some form of credit from it.

3.6 Governance

All components of Basin (i.e., *Well Functions*, *Pumps*, *Well Implementations*, *Aquifers*, and *Aqueducts*) can, but need not, be upgradable. Governance of upgrades to each component is dependent on the owner of the component.

4 Risks

There are risks associated with Basin. This is not an exhaustive list.

The Basin code base is novel. None of its components has been tested in the "real world" prior to its initial deployment. The open source nature of Basin means that others can take advantage of any bugs, flaws or deficiencies in it. While Basin and an initial implementation of each of its components (except *Aqueduct*, which has not been implemented) have been audited,[13,14] it is no guarantee of security.

Particularly before the development of an *Aqueduct*, each component should be verified by users in order to ensure proper functionality.

5 Future Work

Basin is a work in progress. The following are some potential improvements that can be incorporated into Basin through future development:

- Support for ERC-1155[15] and ERC-721[16] Standard tokens can be added.
- Additional *Well Functions* can be implemented.
- A template *Aqueduct* can be implemented.

[13] basin.exchange/06-16-23-halborn-report
[14] basin.exchange/06-16-23-cyfrin-report
[15] ethereum.org/en/developers/docs/standards/tokens/erc-1155
[16] ethereum.org/en/developers/docs/standards/tokens/erc-721