Single-Validator Stake Pool

Trustless liquid staking for all Solana validators.

Information Account Address Single Pool Program SVSPxpvHdN29nkVg9rPapPNDddN5DipNLRUFhyjFThE

Overview

The single-validator stake pool program is an SPL program that enables liquid staking with zero fees, no counterparty, and 100% capital efficiency.

The program defines a canonical pool for every vote account, which can be initialized permissionlessly, and mints tokens in exchange for stake delegated to its designated validator.

The program is a stripped-down adaptation of the existing multi-validator stake pool program, with approximately 80% less code, to minimize execution risk.

Source

The Single Pool Program's source is available onGitHub .

Security Audits

The Single Pool Program has received three audits to ensure total safety of funds:

- Zellic (2024-01-02)* Review commit hashef44df9
-
    - Final reporthttps://github.com/solana-labs/security-audits/blob/master/spl/ZellicSinglePoolAudit-2024-01-02.pdf
- Neodyme (2023-08-08)* Review commit hash735d729
-
    - Final reporthttps://github.com/solana-labs/security-audits/blob/master/spl/NeodymeSinglePoolAudit-2023-08-08.pdf
- Zellic (2023-06-21)* Review commit hash9dbdc3b
-
    - Final reporthttps://github.com/solana-labs/security-audits/blob/master/spl/ZellicSinglePoolAudit-2023-06-21.pdf

Interface

The single-validator stake pool program is written in Rust and available oncrates.io anddocs.rs .

Javascript bindings are available forWeb3.js Classic andWeb3.js Next .

Reference Guide

Environment Setup

- CLI
- WEB3.JS CLASSIC
- WEB3.JS NEXT

The easiest way to interact with the single pool program is using thespl-single-pool command-line program. WithRust installed , run:

cargo install spl-single-pool-cli Runspl-single-pool --help for a full description of available commands.

Configuration

Thespl-single-pool configuration is shared with thesolana command-line tool.

Current Configuration

solana config get Config File: {HOME}/.config/solana/cli/config.yml RPC URL: https://api.mainnet-beta.solana.com WebSocket URL: wss://api.mainnet-beta.solana.com/ (computed) Keypair Path: {HOME}/.config/solana/id.json Commitment: confirmed

Cluster RPC URL

SeeSolana clusters for cluster-specific RPC URLs

solana config set --url https://api.devnet.solana.com

Default Keypair

SeeKeypair conventions for information on how to setup a keypair if you don't already have one.

Keypair File

solana config set --keypair {HOME}/new-keypair.json Hardware Wallet URL (SeeURL spec)

solana config set --keypair usb://ledger/ spl-single-pool generally uses the default keypair as the fee-payer, the wallet to draw funds from (for instance, to fund new stake accounts), and the signing authority on accounts that require one. When token accounts are required, it defaults to the default keypair's associated account. All of these roles can be overridden by command-line flags.

pnpm

pnpm install @solana/spl-single-pool-classic

Yarn

yarn add @solana/spl-single-pool-classic

npm

npm install @solana/spl-single-pool-classic

Configuration

You can connect to different clusters usingConnection in@solana/web3.js

import

{ Connection , clusterApiUrl }

from

'@solana/web3.js' ; const connection =

new

Connection (clusterApiUrl ('devnet') ,

'confirmed') ;

Keypair

You can either get your keypair usingKeypair from@solana/web3.js , or let the user's wallet handle the keypair and usesendTransaction fromwallet-adapter

pnpm

pnpm install @solana/spl-single-pool

Yarn

yarn add @solana/spl-single-pool

npm

npm install @solana/spl-single-pool

Configuration

You can connect to different clusters usingcreateDefaultRpcTransport andcreateSolanaRpc in@solana/web3.js

import

{ createDefaultRpcTransport , createSolanaRpc }

from

'@solana/web3.js' ; const transport =

createDefaultRpcTransport ({ url :

'https://api.devnet.solana.com'

}) ; const rpc =

createSolanaRpc ({ transport }) ;

Keypair

Web3.js Next usescrypto.subtle for key management. For local development, one can create an object{ privateKey, publicKey } usingcrypto.subtle.importKey to produce signing (private) and verifying (public) keys. Generally, however, one should allow the user wallet to manage keys, as the system when used properly is designed to not allow key material to leave secure storage.

Setting up a single-validator pool

Creating the pool

A single-validator stake pool can be created permissionlessly, by anyone, for a given vote account. This allows you to receive the full staking yield you would by staking directly while holding the value in a tokenized form. It also allows you to buy or sell stakes smaller than the minimum delegation on the market.

Assuming a vote accountAmmgaa2iZfA745BmZMhkcS27uh87fEVDC6Gm2RXz5hrC exists, we create a pool at addressDkE6XFGbqSyYzRugLVSmmB42F9BQZ7mZU837e2Cti7kb :

- CLI
- WEB3.JS CLASSIC
- WEB3.JS NEXT

spl-single-pool manage initialize Ammgaa2iZfA745BmZMhkcS27uh87fEVDC6Gm2RXz5hrC const voteAccountAddress =

new

PublicKey ('Ammgaa2iZfA745BmZMhkcS27uh87fEVDC6Gm2RXz5hrC') ; const transaction =

await SinglePoolProgram . initialize (connection , voteAccountAddress , feePayerAddress ,) ;

// sign with the fee payer const voteAccountAddress =

'Ammgaa2iZfA745BmZMhkcS27uh87fEVDC6Gm2RXz5hrC'

as VoteAccountAddress ; const transaction =

await SinglePoolProgram . initialize (rpc , voteAccountAddress , feePayerAddress ,) ;

// sign with the fee payer

Managing token metadata

By default, when a pool is created, it also creates Metaplex token metadata for the mint associated with the pool. If, for whatever reason, this was opted out of by the pool creator, anyone may create the default metadata permissionlessly:

- CLI
- WEB3.JS CLASSIC
- WEB3.JS NEXT

spl-single-pool manage create-token-metadata --pool DkE6XFGbqSyYzRugLVSmmB42F9BQZ7mZU837e2Cti7kb const transaction =

await SinglePoolProgram . createTokenMetadata (poolAddress , feePayerAddress) ;

// sign with the fee payer const transaction =

await SinglePoolProgram . createTokenMetadata (poolAddress , feePayerAddress) ;

// sign with the fee payer The default token metadata is only minimally helpful, spotlighting the address of the validator vote account. The owner of the vote account, however, can change the metadata to anything they wish. They prove their identity by signing with the vote account's authorized withdrawer; this is the only permissioned instruction on the pool.

- CLI
- WEB3.JS CLASSIC
- WEB3.JS NEXT

spl-single-pool manage update-token-metadata DkE6XFGbqSyYzRugLVSmmB42F9BQZ7mZU837e2Cti7kb "My Cool Pool" cPool "https://www.cool.pool/token.jpg" const transaction =

await SinglePoolProgram . updateTokenMetadata (voteAccountAddress , authorizedWithdrawerAddress , 'My Cool Pool' , 'cPool' , 'https://www.cool.pool/token.jpg' ,) ;

// sign with the fee payer and authorized withdrawer const transaction =

await SinglePoolProgram . updateTokenMetadata (voteAccountAddress , authorizedWithdrawerAddress , 'My Cool Pool' , 'cPool' , 'https://www.cool.pool/token.jpg' ,) ;

// sign with the fee payer and authorized withdrawer The URL parameter is optional.

Using a single-validator pool

Depositing

When a pool is created, its stake account is delegated to the appropriate vote account, and for that epoch, stake in an "activating" state can be deposited into it. After this epoch, stake must be in an "active" state to deposit into the pool. That is, it must be delegated to the vote account, and a deposit can only be performed after the next epoch boundary.

Assuming the stake account9cc4cmLcZA89fYmcVPPTLmHPQ5gab3R6jMqj124abkSi is in an active state:

- CLI
- WEB3.JS CLASSIC
- WEB3.JS NEXT

spl-single-pool deposit 9cc4cmLcZA89fYmcVPPTLmHPQ5gab3R6jMqj124abkSi When an explicit stake account address is provided, the CLI can determine the pool address automatically. const transaction =

await SinglePoolProgram . deposit ({ connection , pool : poolAddress , userWallet , userStakeAccount , }) ;

// sign with the fee payer and stake account withdraw authority, if these signers differ // userWallet is a convenience parameter to use one account as a payer, authority, and lamport recipient const transaction =

await SinglePoolProgram . deposit ({ rpc , pool : poolAddress , userWallet , userStakeAccount , }) ;

// sign with the fee payer and stake account withdraw authority, if these signers differ // userWallet is a convenience parameter to use one account as a payer, authority, and lamport recipient All versions of thedeposit command/transaction automatically create the associated token account for the pool token if it doesn't exist and no auxiliary token account address is provided.

The program also makes available a convenience address for each pool, called the default deposit address. This allows a flow where you create and delegate a stake at a program-derived address, and then can deposit this stake after the epoch boundary, without having to generate or keep track of any new keypairs. The user retains full authority on the stake account until they decide to deposit.

- CLI
- WEB3.JS CLASSIC
- WEB3.JS NEXT

spl-single-pool create-default-stake --pool DkE6XFGbqSyYzRugLVSmmB42F9BQZ7mZU837e2Cti7kb 1000000000 Once the stake becomes active, typically in the next epoch:

spl-single-pool deposit --pool DkE6XFGbqSyYzRugLVSmmB42F9BQZ7mZU837e2Cti7kb --default-stake-account const transaction =

await SinglePoolProgram . createAndDelegateUserStake (connection , voteAccountAddress , userWallet , 1000000000 ,) ;

// sign with user wallet Once the stake becomes active, typically in the next epoch:

const transaction =

await SinglePoolProgram . deposit ({ connection , pool : poolAddress , userWallet , depositFromDefaultAccount :

true , }) ;

// sign with user wallet const transaction =

await SinglePoolProgram . createAndDelegateUserStake (rpc , voteAccountAddress , userWallet , 1000000000n ,) ;

// sign with user wallet, which is used as the fee payer and as the base address for a seeded account Once the stake becomes active, typically in the next epoch:

const transaction =

await SinglePoolProgram . deposit ({ rpc , pool : poolAddress , userWallet , depositFromDefaultAccount :

true , }) ;

// sign with user wallet, which is used as the fee payer and as the base address for a seeded account

Withdrawing

Withdrawing is simple, burning tokens to receive the amount of stake they're backed by. Stake can be withdrawn into an active stake account delegated to the appropriate vote account, or to a new stake account, with all authority assigned to the user wallet. Internally, all versions of thewithdraw command/transaction use a token delegate to accomplish the burn. This means the user does not have to provide a wallet signature to the single pool program.

- CLI
- WEB3.JS CLASSIC
- WEB3.JS NEXT

spl-single-pool withdraw --pool DkE6XFGbqSyYzRugLVSmmB42F9BQZ7mZU837e2Cti7kb 1000000000 The--deactivate flag may also be passed, as a convenience to start the undelegation process. const withdrawAccount =

new

Keypair ( ) ; const transaction =

await SinglePoolProgram . withdraw ({ connection , pool : poolAddress , userWallet , userStakeAccount : withdrawAccount . publicKey , tokenAmount :

1000000000 , createStakeAccount :

true , }) ;

// sign with fee payer, and the stake account keypair if a new account is being created const

{ publicKey , privateKey }

=

await

generateKeyPair ( ) ; const transaction =

await SinglePoolProgram . withdraw ({ rpc , pool : poolAddress , userWallet , userStakeAccount : publicKey , tokenAmount :

1000000000n , createStakeAccount :

true , }) ;

// sign with fee payer, and the stake account keypair if a new account is being createdEdit this page Previous Command-line Interface Next Feature Proposal Program * Overview * Source * Security Audits * Interface * Reference Guide * * Environment Setup * * Configuration * * Configuration * * Keypair * * Configuration * * Keypair * * Setting up a single-validator pool * * Using a single-validator pool