

Layr-Labs /
eigenlayer-contracts

<> Code

Issues

30



Pull requests

41



Actions



Projects



Security

eigenlayer-contracts / docs /



wadealexc feat: partial withdrawal batching (#515)



b4fa900 · last month



Name	Name	Last commit date
..		
core	feat: partial withdrawal ba...	last month
experimental	chore: rename to rewards ...	4 months ago
images	feat: partial withdrawal ba...	last month
README.md	feat: partial withdrawal ba...	last month

README.md



EigenLayer Docs - v0.4.0 Release

This repo contains the EigenLayer core contracts, which enable restaking of liquid staking tokens (LSTs) and beacon chain ETH to secure new services, called AVSs (actively validated services). For more info on AVSs, check out the EigenLayer middleware contracts [here](#).

This document provides an overview of system components, contracts, and user roles. Further documentation on the major system contracts can be found in [/core](#).

Contents

- [System Components](#)
 - [EigenPodManager](#)
 - [StrategyManager](#)
 - [DelegationManager](#)

- [RewardsCoordinator](#)
- [AVSDirectory](#)
- [Slasher](#)
- [Roles and Actors](#)
- [Common User Flows](#)
 - [Depositing Into EigenLayer](#)
 - [Delegating to an Operator](#)
 - [Undelegating or Queueing a Withdrawal](#)
 - [Completing a Withdrawal as Shares](#)
 - [Completing a Withdrawal as Tokens](#)
 - [Withdrawal Processing: Validator Exits](#)
 - [Withdrawal Processing: Partial Beacon Chain Withdrawals](#)

System Components

EigenPodManager

File	Type	Proxy
EigenPodManager.sol	Singleton	Transparent proxy
EigenPod.sol	Instanced, deployed per-user	Beacon proxy

These contracts work together to enable native ETH restaking:

- Users deploy `EigenPods` via the `EigenPodManager`, which contain beacon chain state proof logic used to verify a validator's withdrawal credentials and current balances. An `EigenPod`'s main role is to serve as the fee recipient and/or withdrawal credentials for one or more of a user's validators.
- The `EigenPodManager` handles `EigenPod` creation and accounting+interactions between users with restaked native ETH and the `DelegationManager`.

See full documentation in:

- [/core/EigenPodManager.md](#)
- [/core/EigenPod.md](#)

StrategyManager

File	Type	Proxy
StrategyManager.sol	Singleton	Transparent proxy
StrategyFactory.sol	Singleton	Transparent proxy
StrategyBaseTVLLimits.sol	Instanced, one per supported token	<ul style="list-style-type: none">- Strategies deployed outside the StrategyFactory use transparent proxies- Anything deployed via the StrategyFactory uses a Beacon proxy

These contracts work together to enable restaking for ERC20 tokens supported by EigenLayer:

- The `StrategyManager` acts as the entry and exit point for any supported tokens in EigenLayer. It handles deposits into LST-specific strategies, and manages accounting+interactions between users with restaked LSTs and the `DelegationManager`.
- `StrategyFactory` allows anyone to deploy strategies to support deposits/withdrawals for new ERC20 tokens
- `StrategyBaseTVLLimits` is deployed as multiple separate instances, one for each supported token. When a user deposits into a strategy through the `StrategyManager`, this contract receives the tokens and awards the user with a proportional quantity of shares in the strategy. When a user withdraws, the strategy contract sends the LSTs back to the user.

See full documentation in [/core/StrategyManager.md](#).

DelegationManager

File	Type	Proxy
DelegationManager.sol	Singleton	Transparent proxy

The `DelegationManager` sits between the `EigenPodManager` and `StrategyManager` to manage delegation and undelegation of Stakers to Operators. Its primary features are to allow Operators to register as Operators (`registerAsOperator`), to keep track of shares being delegated to Operators across different strategies, and to manage withdrawals on behalf of the `EigenPodManager` and `StrategyManager`.

See full documentation in </core/DelegationManager.md> .

RewardsCoordinator

File	Type	Proxy
RewardsCoordinator.sol	Singleton	Transparent proxy

The `RewardsCoordinator` is the main entry point of submission and claiming of ERC20 rewards in EigenLayer. It carries out three basic functions:

- AVSs (via the AVS's contracts) submit "rewards submissions" to their registered Operators and Stakers over a specific time period
- *Off-chain*, the rewards updater will use each RewardsSubmission time period to apply reward amounts to historical Staker/Operator stake weights. This is consolidated into a merkle root that is posted *on-chain* to the RewardsCoordinator , allowing Stakers/Operators to claim their allocated rewards.
- Stakers/Operators can claim rewards posted by the rewards updater.

See full documentation in </core/RewardsCoordinator.md> .

AVSDirectory

File	Type	Proxy
AVSDirectory.sol	Singleton	Transparent proxy

The `AVSDirectory` handles interactions between AVSs and the EigenLayer core contracts. Once registered as an Operator in EigenLayer core (via the `DelegationManager`), Operators can register with one or more AVSs (via the AVS's contracts) to begin providing services to them offchain. As a part of registering with an AVS, the AVS will record this registration in the core contracts by calling into the `AVSDirectory` .



See full documentation in </core/AVSDirectory.md> .

For more information on AVS contracts, see the [middleware repo](#).

Slasher

File	Type	Proxy
------	------	-------

File	Type	Proxy
Slasher.sol	-	-

 The Slasher contract is under active development and its interface expected to change. We recommend writing slashing logic without integrating with the Slasher at this point in time. Although the Slasher is deployed, it will remain completely paused/unusable during M2. No contracts interact with it, and its design is not finalized. 

Roles and Actors

To see an example of the user flows described in this section, check out our integration tests: </src/test/integration>.

Staker

A Staker is any party who has assets deposited (or "restaked") into EigenLayer. Currently, these assets can be:

- Native beacon chain ETH (via the EigenPodManager)
- Liquid staking tokens (via the StrategyManager): cbETH, rETH, stETH, ankrETH, OETH, osETH, swETH, wBETH

Stakers can restake any combination of these: a Staker may hold ALL of these assets, or only one of them.

Flows:

- Stakers **deposit** assets into EigenLayer via either the StrategyManager (for LSTs) or EigenPodManager (for beacon chain ETH)
- Stakers **withdraw** assets via the DelegationManager, *no matter what assets they're withdrawing*
- Stakers **delegate** to an Operator via the DelegationManager

Unimplemented as of v0.4.0:

- Stakers are at risk of being slashed if the Operator misbehaves

Operator

An Operator is a user who helps run the software built on top of EigenLayer (AVSs). Operators register in EigenLayer and allow Stakers to delegate to them, then opt in to provide various services built on top of EigenLayer. Operators may themselves be Stakers; these are not mutually exclusive.

Flows:

- User can **register** as an Operator via the DelegationManager
- Operators can **deposit** and **withdraw** assets just like Stakers can
- Operators can opt in to providing services for an AVS using that AVS's middleware contracts. See the [EigenLayer middleware](#) repo for more details.

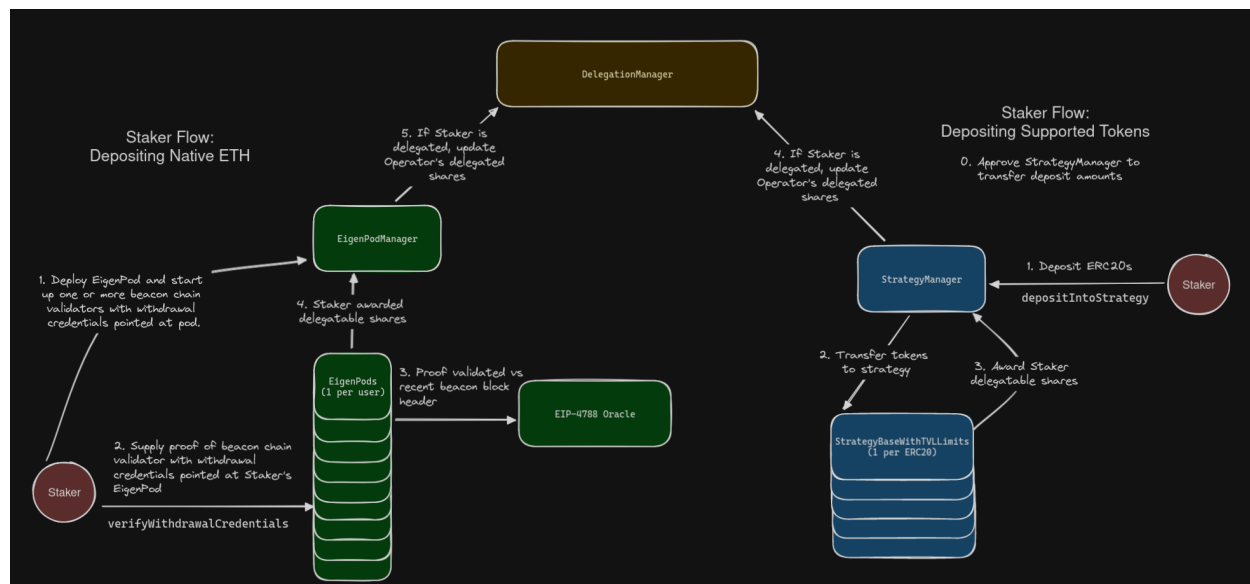
Unimplemented as of v0.4.0:

- Operators may be slashed by the services they register with (if they misbehave)

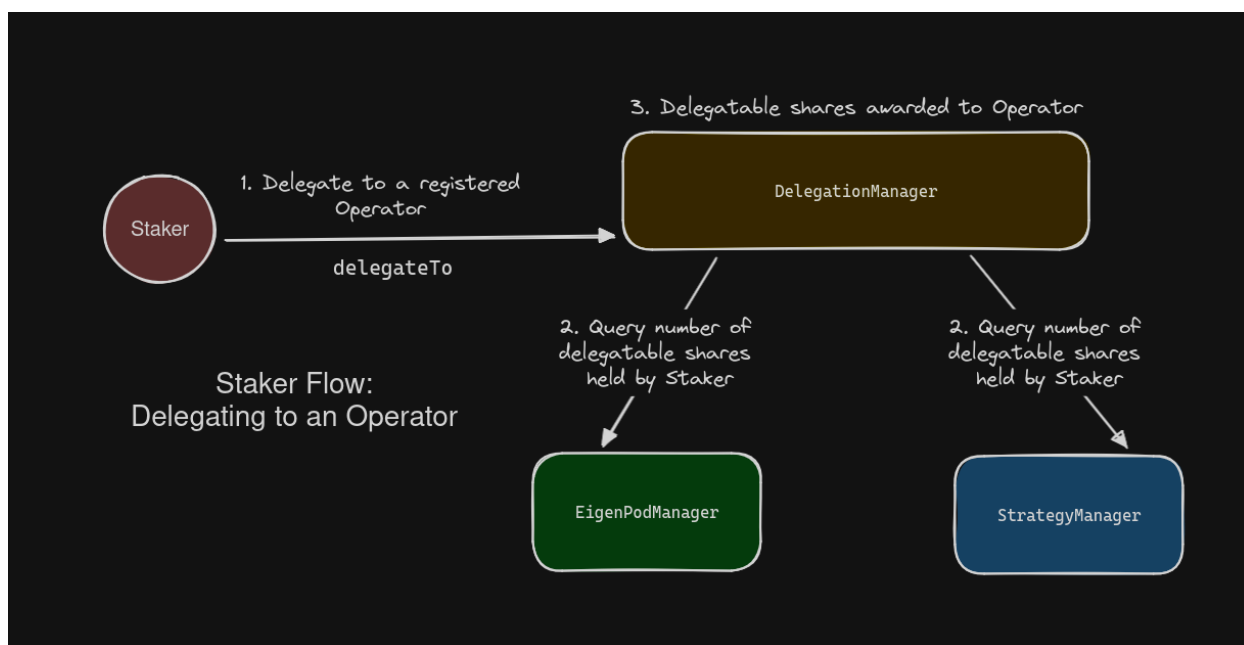
Common User Flows

Depositing Into EigenLayer

Depositing into EigenLayer varies depending on whether the Staker is depositing Native ETH or LSTs:

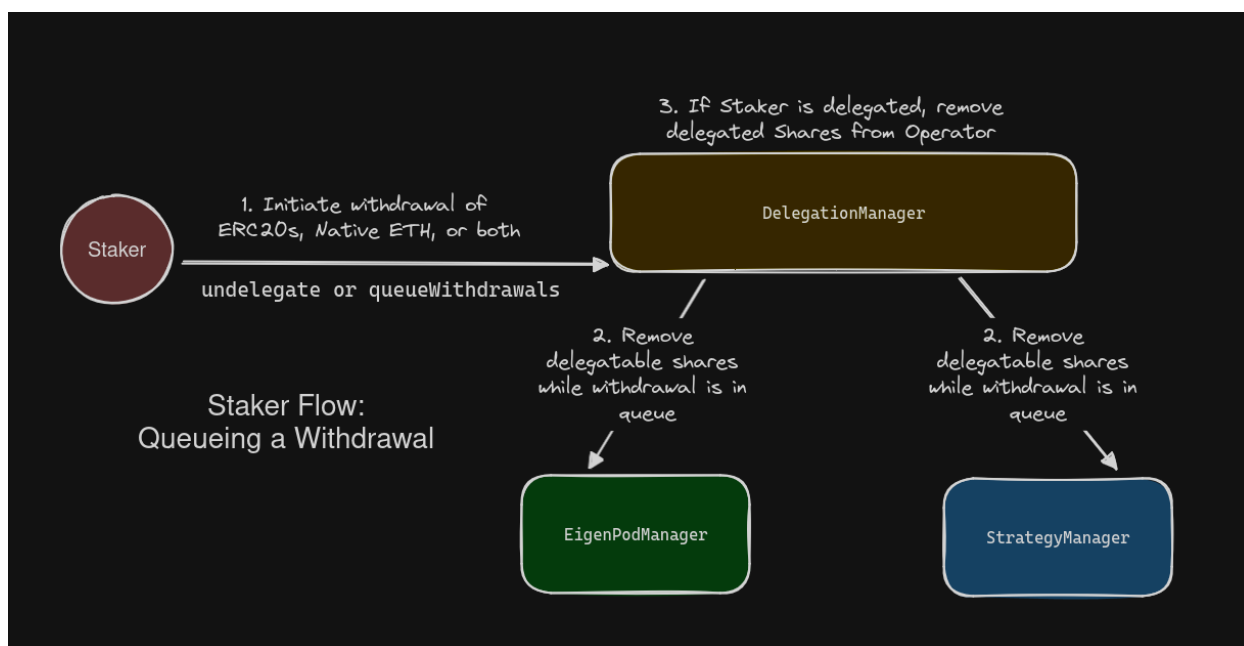


Delegating to an Operator



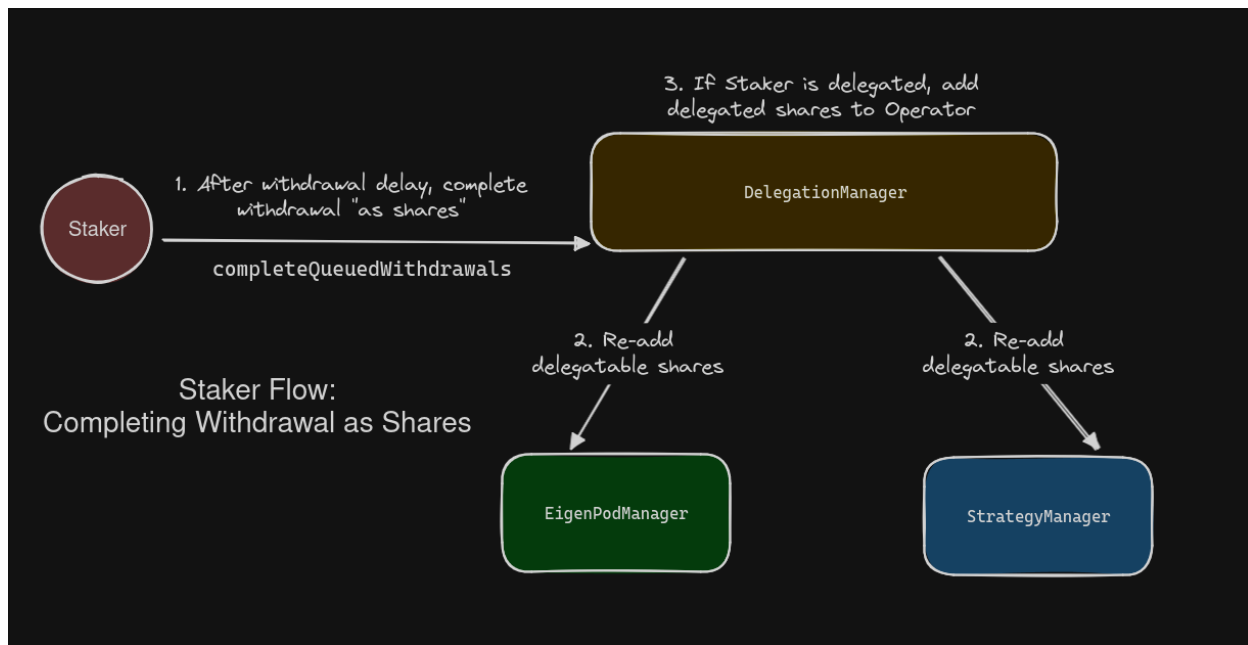
Undelegating or Queueing a Withdrawal

Undelegating from an Operator automatically queues a withdrawal that needs to go through the **DelegationManager's** withdrawal delay. Stakers that want to withdraw can choose to `undelegate`, or can simply call `queueWithdrawals` directly.



Completing a Withdrawal as Shares

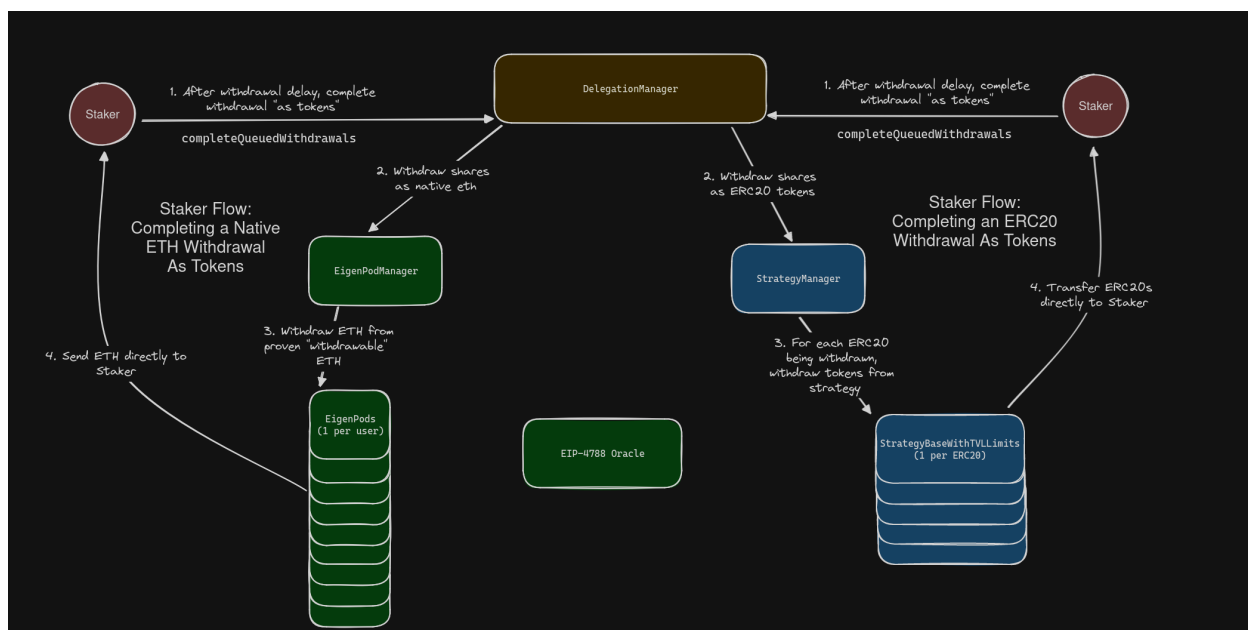
This flow is mostly useful if a Staker wants to change which Operator they are delegated to. The Staker first needs to undelegate (see above). At this point, they can delegate to a different Operator. However, the new Operator will only be awarded shares once the Staker completes their queued withdrawal "as shares":



Completing a Withdrawal as Tokens

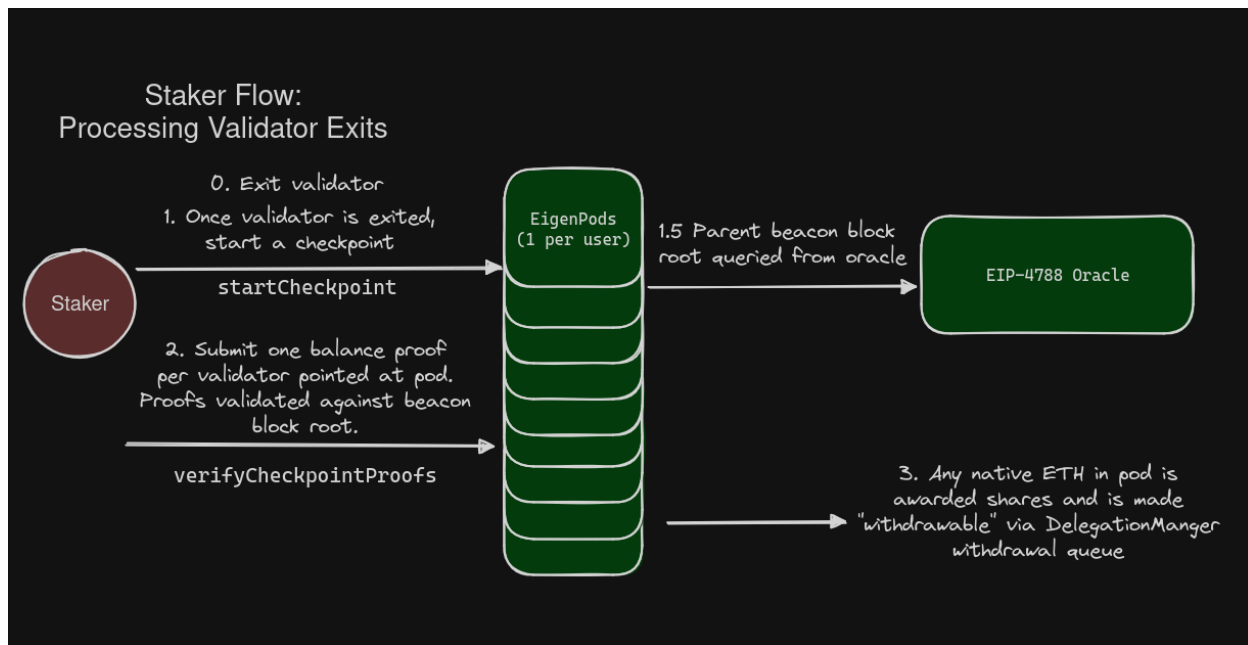
Completing a queued withdrawal as tokens is roughly the same for both native ETH and LSTs.

However, note that *before* a withdrawal can be completed, native ETH stakers will need to perform additional steps, detailed in the diagrams below.



EigenPods : Processing Validator Exits

If a Staker wants to fully withdraw from the beacon chain, they need to perform these additional steps before their withdrawal is completable:



EigenPods : Processing Validator Yield

As the Staker's EigenPod accumulates consensus layer or execution layer yield, the EigenPod's balance will increase. The Staker can Checkpoint their validator to claim this yield as shares, which can either remain staked in EigenLayer or be withdrawn via the DelegationManager withdrawal queue:

