Community Staker / Operator Actions

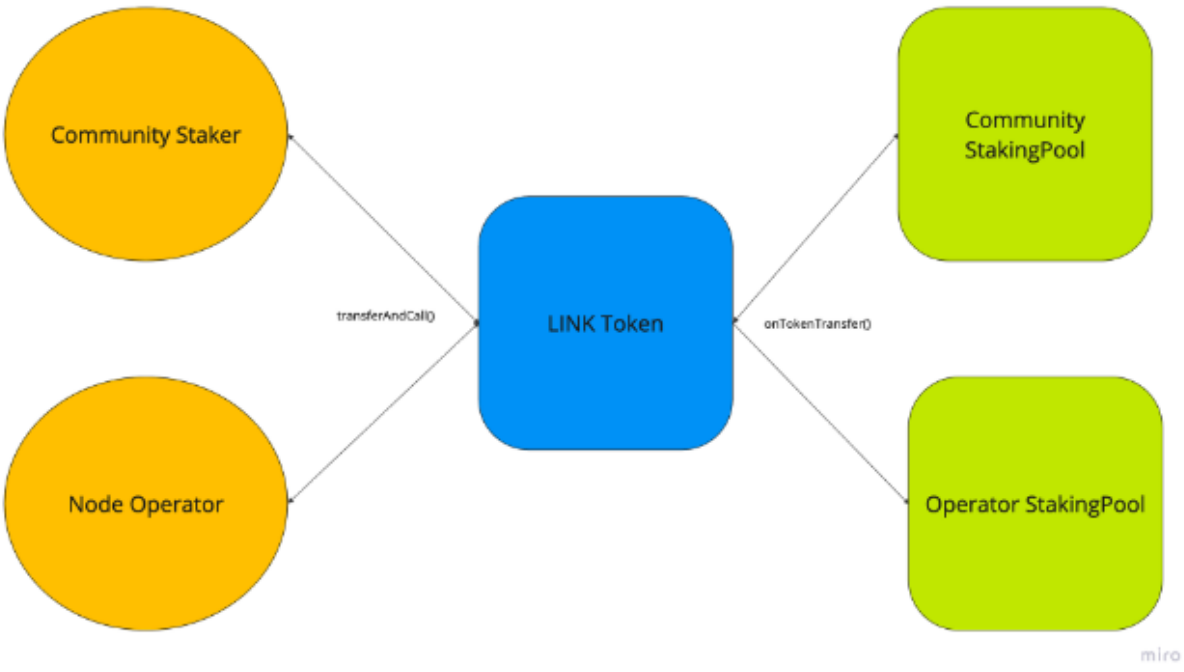
Table of Contents - Community Staker / Operator Actions

Motivation

This document outlines flows and actions for Community Stakers and Operator Stakers.

NOTE: This documentation and code contains parameter config values. These config values are for illustration purposes only in order to explain how the code executes through examples. Such configs will be set at different values upon launch.

Staking

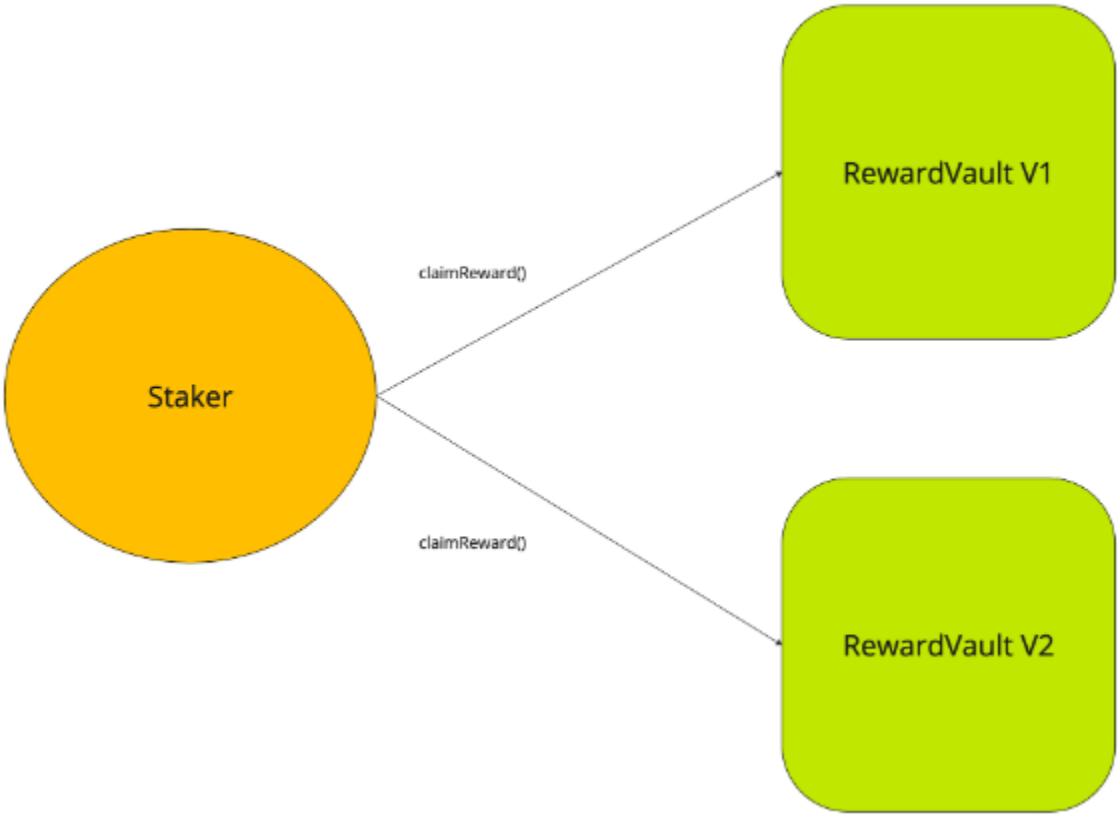


- Staker calls `transferAndCall` on the LINK Token contract.
    - Staker specifies the staking pool's address. Community Stakers will specify the address of the `CommunityStakingPool` and Node Operator Stakers will specify the address of the `OperatorStakingPool`.
    - Staker specifies the `amount` of LINK tokens in juels that they want to stake
- LINKToken calls `onTokenTransfer` on the `StakingPool`. The `StakingPool` will
    - Verify that `amount` is within the staking limits
    - Verify that the `staker` can stake in the pool that they are trying to stake in. For example, Node Operator Stakers should not be able to stake in the `CommunityStakingPool` and Community Stakers should not be able to stake in the `OperatorStakingPool`
- The staker's LINK tokens move from their wallet to the corresponding `StakingPool`.

Unstaking

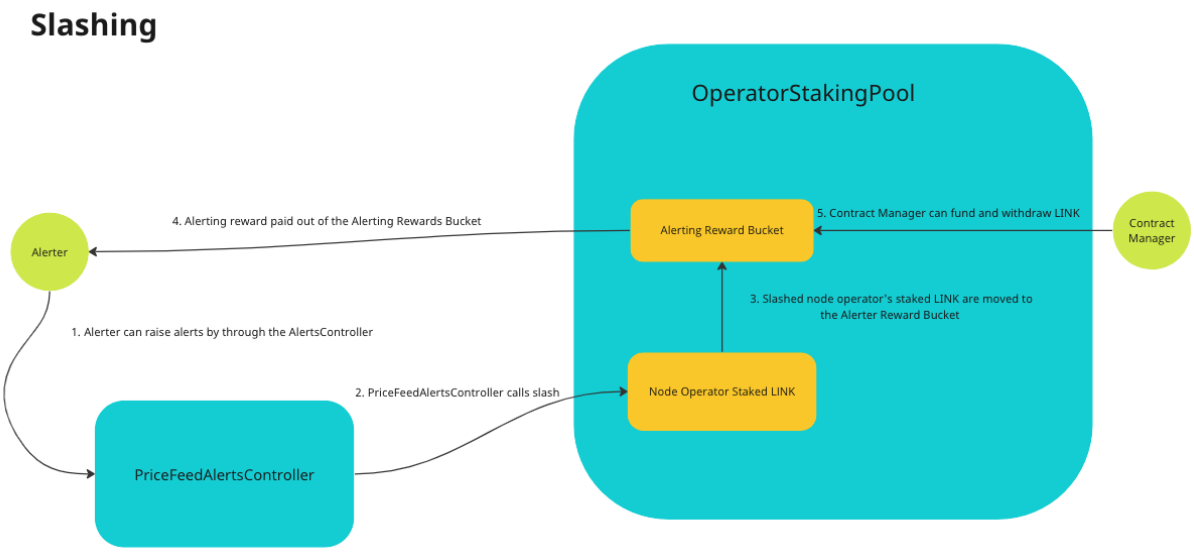
- Staker calls `unbond` on the `StakingPool` contract
- They can call `unstake` after the unbonding period and during the claim period to unstake their LINK tokens on the `StakingPool` page.
    - Staker can pass in an `amount` field to specify how much of their staked LINK to unstake.
    - Staker can pass in a boolean to indicate whether or not they want to withdraw rewards as well in the same transaction.
    - The Staker's earned reward is stored based on the current multiplier applied.
    - The Staker's multiplier is fully reset to 0.
- The Staker's ability to withdraw goes away after the claim period following the unbonding period is over, and they will need to restart the unbonding period again by calling `unbond`.

Claiming Rewards



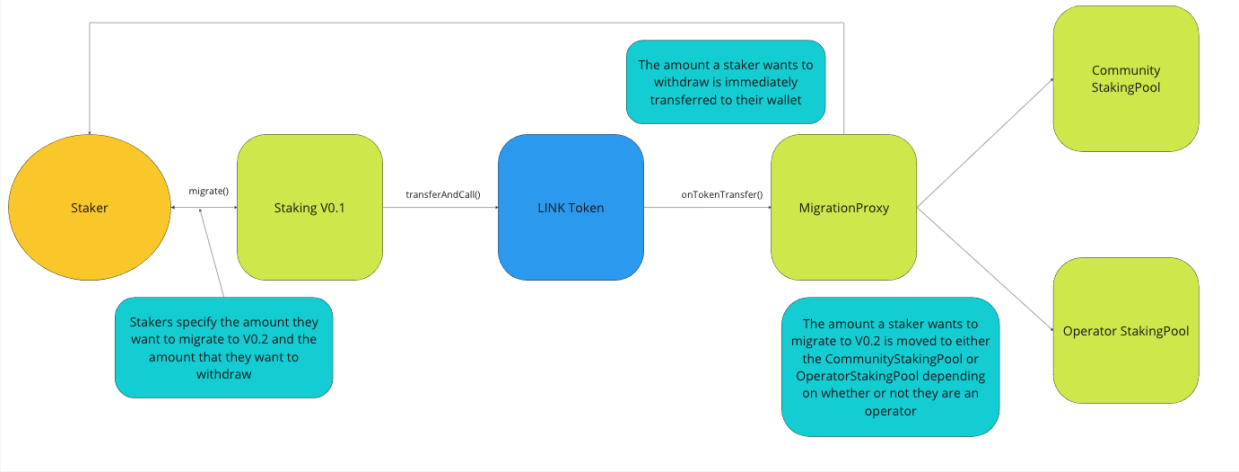
- Staker directly `claimRewards` on the `RewardVault` contract to claim any claimable rewards in that vault
    - Rewards are claimed in full (no partial amounts.)
    - `RewardVault` calculates the staker's rewards whilst taking into account the staker's multiplier.
    - LINK tokens move from the `RewardVault` to the staker's wallet.

Alerting



- Alerter calls `raiseAlert` on the `AlertsController`, which then calls `slash` in the `OperatorStakingPool`.
- The staked LINK slashed from affected node operators are moved to the `Alerting Reward Bucket` within the `OperatorStakingPool`.
- The `Alerting Reward Bucket` automatically transfers LINK rewards to the `Alerter`.

Migrating From v0.1 to v0.2



- Prerequisites
    - `MigrationProxy` and StakingPools are deployed.
    - Operators are added to the `OperatorStakingPool`.
    - The Merkle root has been set in the `CommunityStakingPool`.
    - Both pools are started by the contract manager calling `start`.
        - The `OperatorStakingPool` cannot be started if there are not enough operators
        - The `CommunityStakingPool` cannot be started if there is no Merkle Root.
    - Set the migration target in the Staking V0.1 contract to the `RewardVault`
- Staker calls `migrate` in the `StakingV0.1` contract. They can pass in the amount they want to unstake/withdraw through the `data` parameter in the `migrate` function.
- Staking v0.1 calls `onTokenTransfer` in the `RewardVault` contract.
- The `MigrationProxy` contract determines if the sender is a Node Operator Staker or Community Staker and routes the migrated LINK to either the `NOPStakingPool` or `CommunityStakingPool`. Any unmigrated LINK tokens are transferred back to the staker.
- LINK flows from Staking v0.1 to the `MigrationProxy` and then to the respective pool.