[

V3migration_banner2 (1)

1920×845 112 KB

](https://europe1.discourse-cdn.com/business20/uploads/aave/original/2X/f/f21bbfde9bf88e7dec852a33bfc5b8e1b6a521f1.jpeg)

# TL;DR

We present to the community the Aave v2 → v3 migration tool, released in parallel with Aave v3 Ethereum, and also covering Aave Polygon and Avalanche.

# The problem

Given that currently Aave v2 and Aave v3 instances are completely isolated from each other, we evaluated that the experience of manually closing the position on v2 and creating the same on v3 was not optimal, burden and cost-wise (gas).

To improve this, together with the more infrastructural work of v3.0.1 development (running on Aave v3 Ethereum), we decided to develop a v2→v3 migration tool, including smart contracts and UI components.

# v2 → v3 migration tool

### Smart contracts

Usually, user experience starts with smart contracts, and this case is no exception. In order to enable the migration feature, we built a MigrationHelper contract working as follows:

- The contract is linked to 2 pool instances: one of Aave v2 as the origin, and another of Aave v3 as the destination.
- If the user only has deposits on Aave v2

(no borrowings), a function migrate()

needs to be called, passing as parameters the information of the assets to migrate, together with signatures to give allowance to the contract in order to perform the migration.

The flow is as follows:

- Withdraw the deposits on behalf of the user from Aave v2 and deposit them on Aave v3.
- Additionally doing an action just before depositing on Aave v3, like wrapping stETH into wstETH on Aave v3 Ethereum.
- Withdraw the deposits on behalf of the user from Aave v2 and deposit them on Aave v3.
- Additionally doing an action just before depositing on Aave v3, like wrapping stETH into wstETH on Aave v3 Ethereum.
- If the user has both deposits and borrowings on Aave v2

, the same migrate()

function needs to be called, passing to the contract in addition to the signatures deposits' allowance, signatures of credit delegation of Aave v3.

Then, the contract will:

- Flash loan funds from Aave v3 to close the borrowings of the user on Aave v2.
- Withdraw the deposits from Aave v2 and deposit them on behalf of the user on Aave v3.
- By using the credit delegation signed by the user, keep the funds "flashed" from Aave v3 as borrowings of the user there, factually migrating his debt from v2 to v3.
- Flash loan funds from Aave v3 to close the borrowings of the user on Aave v2.
- Withdraw the deposits from Aave v2 and deposit them on behalf of the user on Aave v3.

- By using the credit delegation signed by the user, keep the funds “flashed” from Aave v3 as borrowings of the user there, factually migrating his debt from v2 to v3.

Additional characteristics of this contract are:

- In order to have a really simple flow and calculations, the migration contract currently works with “whole” amounts: the user can choose which assets to migrate, but he should migrate the full amount of each asset, no matter if deposits or borrowings.

This decision was taken because security-wise the system becomes solid by default, given there are not really asymmetries.

- The contract is designed to be agnostic of “what is better for the user”: if the Aave v3 protocol allows executing the migration, the position will be migrated. This means the security of Aave v3 is fully inherited. All additional ad-hoc considerations are moved off-chain, to the UI libraries’ components.

For example, it is simply impossible for the migrator contract to open and migrate a position and create a situation of HF < 1 during the process because the Aave protocol will simply not allow it.

- The contract is generally permissionless, only with a function to rescue tokens sent by error to it (controlled by the Aave governance), as in general, the contract doesn’t create any meaningful “leftovers”.

- The contract is network agnostic (supporting Ethereum, Polygon, and Avalanche), but abstract enough to add additional logic in a simple way, like in the case of Ethereum, doing wrapping of stETH to wstETH as an intermediate step in some cases.

- The usage of signatures (permits and credit delegation), allows having only 1 transaction to perform this type of migration, no matter the assets the user wants to migrate.

## User interface

As the user interface on app.aave.com is one of the most used to access the Aave protocol, even if owned by Aave Companies, we decided that the way we can give more value to the community is to develop a migration view on it.

As we commented previously, it is on the UI/off-chain utility side where the more “granular” logic regarding user positions lives. This reflects on the user experience in the following way:

- The system assumes each asset to be migrated is available in both the origin Aave instance (v2) and destination (v3). More complex cases like having USDC.e on Aave v2 Avalanche and “native” USDC on Aave v3 Avalanche are not covered at the moment by the tool, as they are completely different assets.

- Using the data read from the on-chain contracts, the interface shows the Aave position available to migrate, on the currently connected wallet.

[

migration-example

2356×1362 328 KB

](https://europe1.discourse-cdn.com/business20/uploads/aave/original/2X/d/df002dbe097c1983cd6c642b5a9747ab51af1ce9.png)

Example of an arbitrary user of Aave v2

- The system understands the differences in configurations between v2 and v3, including v3 supply and borrow caps, available liquidity on v3 for the migration, potentially frozen reserves, etc.

- If the user already has a position on Aave v3, it will respect his eMode configuration.

- If one (or more) of the assets to migrate is only enabled in isolation on Aave v3, the system will recommend migrating first the non-isolated ones, but giving to the user the opportunity to migrate the isolated asset (understanding others will be disabled as collateral).

- The interface is also transparently managing scenarios of assets equivalent in practice, for example, stETH and wstETH on Aave v2/v3 Ethereum.

In addition to the specific aspect of the migration tooling, during the process we also improved other parts of the application that required some refactoring:

- Management of multiple signatures for a single transaction (e.g. batch of permit()).

- Management of different types of signatures for a single transaction (e.g. aToken permit() + credit delegation on

vToken).

- Improvement on the generation of signature data for signing.
- Better organization of the assets data, to allow easy migration to Zustand architecture if required in the future, from React.Context

.

## Next steps

Currently, the tool is functional and covers all the assets that don’t have any special dynamics, so the tool should not require much of our intervention in the future.

We keep monitoring the migrations, which are the moment are going smoothly, with approximately $80’000’000 deposits and $17’000’000 borrowings across Ethereum, Avalanche, and Polygon.