



Aave Protocol v2

This repository contains the smart contracts source code and markets configuration for Aave Protocol V2. The repository uses Docker Compose and Hardhat as development environment for compilation, testing and deployment tasks.

What is Aave?

Aave is a decentralized non-custodial liquidity markets protocol where users can participate as depositors or borrowers. Depositors provide liquidity to the market to earn a passive income, while borrowers are able to borrow in an overcollateralized (perpetually) or undercollateralized (one-block liquidity) fashion.

Documentation

The documentation of Aave V2 is in the following [Aave V2 documentation link](#). At the documentation you can learn more about the protocol, see the contract interfaces, integration guides and audits.

For getting the latest contracts addresses, please check the [Deployed contracts page](#) at the documentation to stay up to date.

A more detailed and technical description of the protocol can be found in this repository, [here](#)

Audits

- MixBytes (16/09/2020 - 03/12/2020): report
- PeckShield (29/09/2020 - 03/12/2020) : report (Also available in Chinese in the same folder)
- CertiK (28/09/2020 - 02/12/2020): report
- Consensys Diligence (09/09/2020 - 09/10/2020): report
- Certora, formal verification (02/08/2020 - 29/10/2020): report
- SigmaPrime (January 2021): report

Connect with the community

You can join at the Discord channel or at the Governance Forum for asking questions about the protocol or talk about Aave with other peers.

Getting Started

You can install `@aave/protocol-v2` as an NPM package in your Hardhat, Buidler or Truffle project to import the contracts and interfaces:

```
npm install @aave/protocol-v2
```

Import at Solidity files:

```
1 import {ILendingPool} from "@aave/protocol-v2/contracts/interfaces/ILendingPool.sol";
2
3 contract Misc {
4
5     function deposit(address pool, address token, address user, uint256 amount) public {
6         ILendingPool(pool).deposit(token, amount, user, 0);
7         {...}
8     }
9 }
```

The JSON artifacts with the ABI and Bytecode are also included into the bundled NPM package at `artifacts/` directory.

Import JSON file via Node JS `require`:

```
1 const LendingPoolV2Artifact = require('@aave/protocol-v2/artifacts/contracts/protocol/lendingpool/LendingPool.sol/LendingPool.json');
2
3 // Log the ABI into console
4 console.log(LendingPoolV2Artifact.abi)
```

Setup

The repository uses Docker Compose to manage sensitive keys and load the configuration. Prior any action like test or deploy, you must run `docker-compose up` to start the `contracts-env` container, and then connect to the container console via `docker-compose exec contracts-env bash`.

Follow the next steps to setup the repository:

- Install `docker` and `docker-compose`
- Create an environment file named `.env` and fill the next environment variables

```
1 # Mnemonic, only first address will be used
2 MNEMONIC=""
3
4 # Add Alchemy or Infura provider keys, alchemy takes preference at the
  config level
5 ALCHEMY_KEY=""
6 INFURA_KEY=""
7
8
9 # Optional Etherscan key, for automatize the verification of the
  contracts at Etherscan
10 ETHERSCAN_KEY=""
11
12 # Optional, if you plan to use Tenderly scripts
13 TENDERLY_PROJECT=""
14 TENDERLY_USERNAME=""
```

Markets configuration

The configurations related with the Aave Markets are located at `markets` directory. You can follow the `IAaveConfiguration` interface to create new Markets configuration or extend the current Aave configuration.

Each market should have his own Market configuration file, and their own set of deployment tasks, using the Aave market config and tasks as a reference.

Test

You can run the full test suite with the following commands:

```
1 # In one terminal
2 docker-compose up
```

```
3
4 # Open another tab or terminal
5 docker-compose exec contracts-env bash
6
7 # A new Bash terminal is prompted, connected to the container
8 npm run test
```

Deployments

For deploying Aave Protocol V2, you can use the available scripts located at [package.json](#). For a complete list, run `npm run` to see all the tasks.

Kovan deployment

```
1 # In one terminal
2 docker-compose up
3
4 # Open another tab or terminal
5 docker-compose exec contracts-env bash
6
7 # A new Bash terminal is prompted, connected to the container
8 npm run aave:kovan:full:migration
```

Mainnet fork deployment

You can deploy Aave Protocol v2 in a forked Mainnet chain using Hardhat built-in fork feature:

```
1 docker-compose run contracts-env npm run aave:fork:main
```

Deploy Aave into a Mainnet Fork via console

You can deploy Aave into the Hardhat console in fork mode, to interact with the protocol inside the fork or for testing purposes.

Run the console in Mainnet fork mode:

```
1 docker-compose run contracts-env npm run console:fork
```

At the Hardhat console, interact with the Aave protocol in Mainnet fork mode:

```
1 // Deploy the Aave protocol in fork mode
2 await run('aave:mainnet')
```

```

3
4 // Or your custom Hardhat task
5 await run('your-custom-task');
6
7 // After you initialize the HRE via 'set-DRE' task, you can import any
  TS/JS file
8 run('set-DRE');
9
10 // Import contract getters to retrieve an Ethers.js Contract instance
11 const contractGetters = require('./helpers/contracts-getters'); //
   Import a TS/JS file
12
13 // Lending pool instance
14 const lendingPool = await contractGetters.getLendingPool("LendingPool
   address from 'aave:mainnet' task");
15
16 // You can impersonate any Ethereum address
17 await network.provider.request({ method: "hardhat_impersonateAccount",
   params: ["0xb1adceddb2941033a090dd166a462fe1c2029484"]});
18
19 const signer = await ethers.provider.getSigner("0
   xb1adceddb2941033a090dd166a462fe1c2029484")
20
21 // ERC20 token DAI Mainnet instance
22 const DAI = await contractGetters.getIErc20Detailed("0
   x6B175474E89094C44Da98b954EedeAC495271d0F");
23
24 // Approve 100 DAI to LendingPool address
25 await DAI.connect(signer).approve(lendingPool.address, ethers.utils.
   parseUnits('100'));
26
27 // Deposit 100 DAI
28 await lendingPool.connect(signer).deposit(DAI.address, ethers.utils.
   parseUnits('100'), await signer.getAddress(), '0');

```

Interact with Aave in Mainnet via console

You can interact with Aave at Mainnet network using the Hardhat console, in the scenario where the frontend is down or you want to interact directly. You can check the deployed addresses at <https://docs.aave.com/developers/deployed-contracts>.

Run the Hardhat console pointing to the Mainnet network:

```
1 docker-compose run contracts-env npx hardhat --network main console
```

At the Hardhat console, you can interact with the protocol:

```
1 // Load the HRE into helpers to access signers
```

```
2 run("set-DRE")
3
4 // Import getters to instance any Aave contract
5 const contractGetters = require('./helpers/contracts-getters');
6
7 // Load the first signer
8 const signer = await contractGetters.getFirstSigner();
9
10 // Lending pool instance
11 const lendingPool = await contractGetters.getLendingPool("0
    x7d2768dE32b0b80b7a3454c06BdAc94A69DDc7A9");
12
13 // ERC20 token DAI Mainnet instance
14 const DAI = await contractGetters.getIERc20Detailed("0
    x6B175474E89094C44Da98b954EedeAC495271d0F");
15
16 // Approve 100 DAI to LendingPool address
17 await DAI.connect(signer).approve(lendingPool.address, ethers.utils.
    parseUnits('100'));
18
19 // Deposit 100 DAI
20 await lendingPool.connect(signer).deposit(DAI.address, ethers.utils.
    parseUnits('100'), await signer.getAddress(), '0');
```