

# Importing into Remix

---

Use Firefox browser. The compiler in Remix sometimes crashes in other browsers

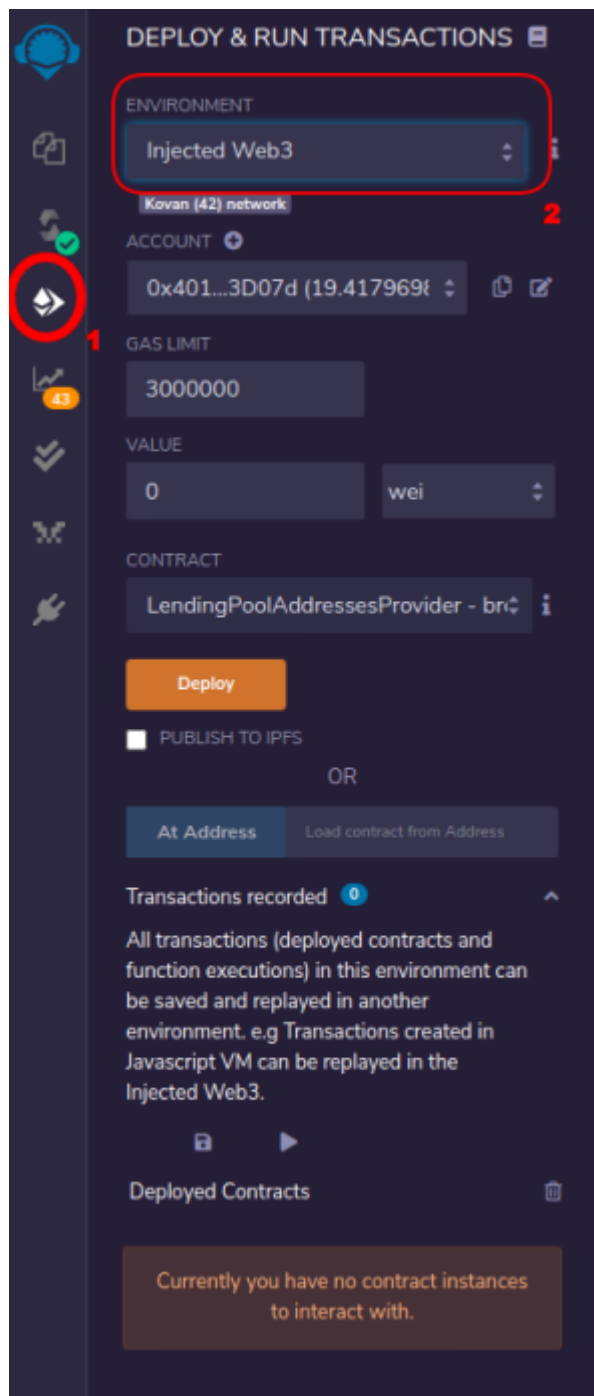
1. You'll need to create a blank file in Remix. This allows you to add directories
2. Create all the top level directories and treat the node dependencies like openzeppelin-solidity as a top level contract
3. Add the subdirectories. At this point you shouldn't have added any code.
4. Add the files and copy in the code.
5. Remember to change the openzeppelin-solidity import references to use relative paths.
6. Compile each file to check that you've imported correctly.
  - >pro tip: when you get an error about a contract having already been declared, look at the import above the offending line. Go to that file. One of the imports in this file has an incorrect path

## Deploying to Kovan from remix

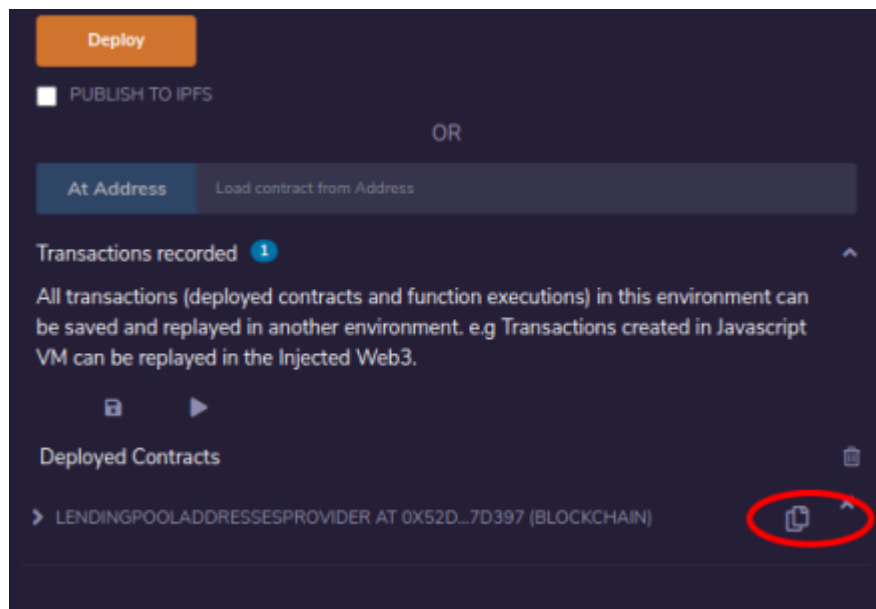
---

The first step to deploying is to find the innermost contract in the Aave architecture. In Aave there's a contract registry that is deployed once. All the contracts reference this registry contract to retrieve the correct versions of their dependencies.

To deploy, click on the deploy menu option (1) and set Environment to Injected Web3 (2). This will prompt your metamask to connect to Remix.



Deploy LendingPoolAddressesProvider.sol:  
Your deployed contract will be displayed in a list.



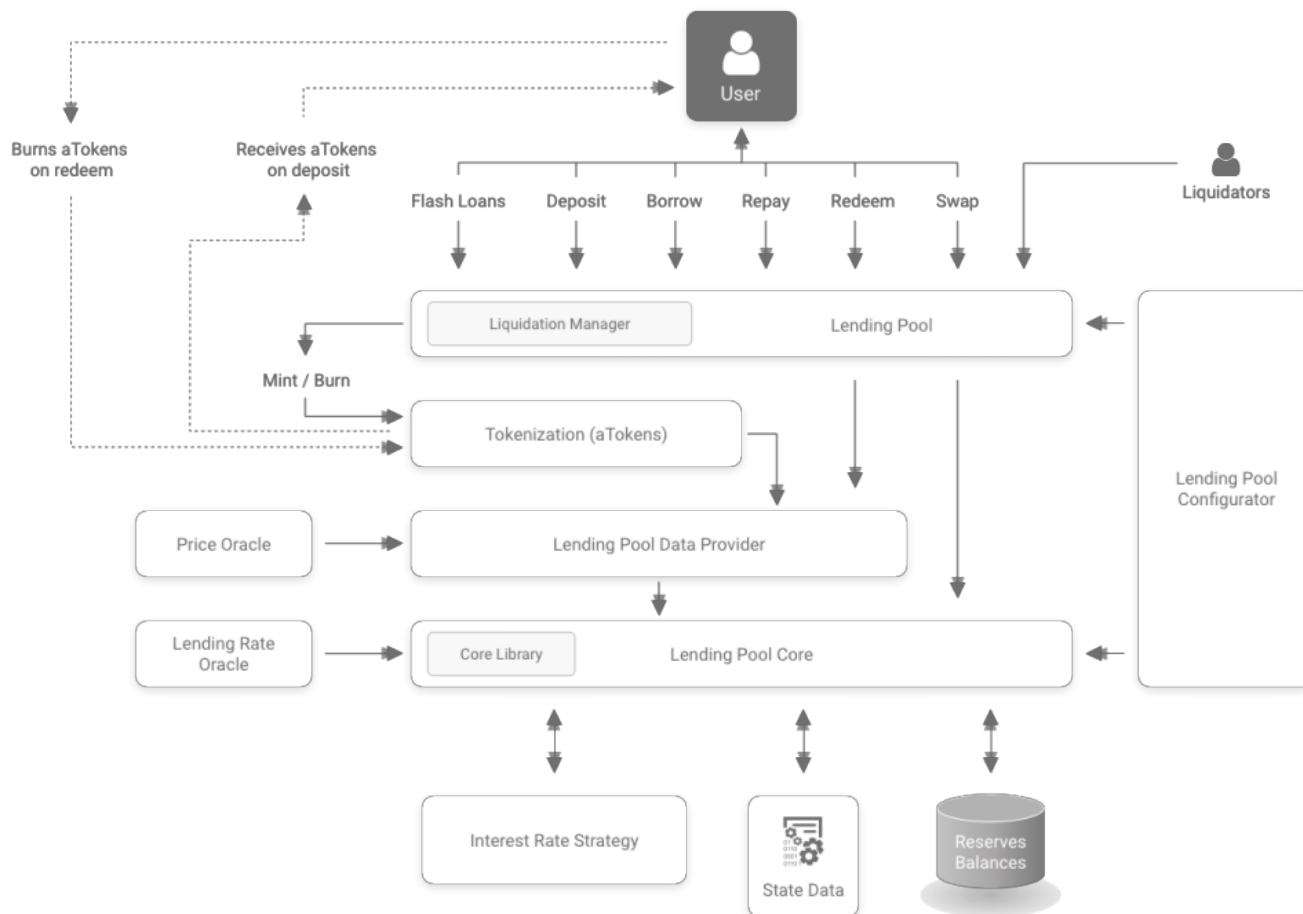
Click the copy icon to retrieve the address. Don't worry about ABIs. The repository has them already.

**Note: all deployments happening from 0x4011Ba13ACD2E078B0234C125D04e5C61803D07d. Since many of these contracts are owned, ownership transfer will have to occur at some point.**

```
LendingPoolAddressesProvider Kovan address:  
0x1a84C655a3F178BeA02DcBc9F46Bc84c17c36FC0
```

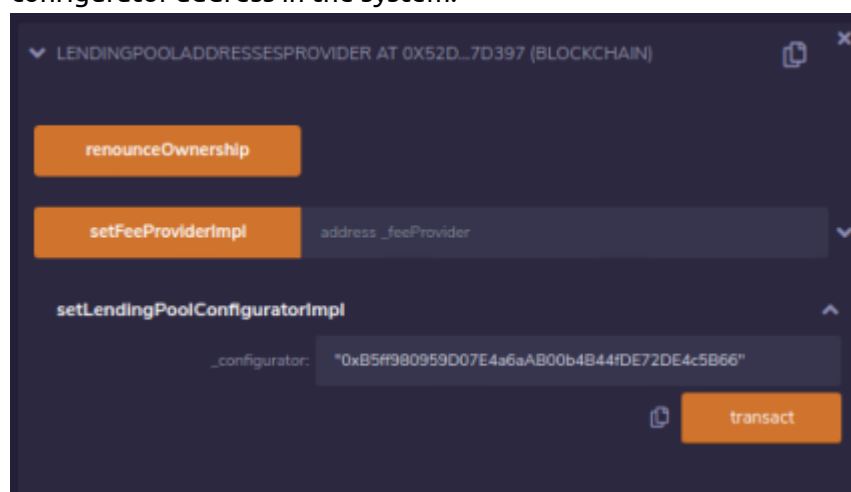
Next we should look at all the expected contracts in LendingpoolAddressesProvider and deploy them and their dependencies.

We'd have to start with their dependencies first. The best dependencies to start with are those that have none. The diagram below is the core architecture of the Aave smart contracts:



let's start the LendingPoolConfigurator. This contract was written before EIP-170 which sets an upper limit on contract size. To get around this, we set Remix to optimize compile.

We initially get an address for LendingPoolConfigurator. We inform LendingPoolAddress provider of its existence by calling the set function for Configurator. This spawns a new proxy contract. If we call LendingPoolAddressProvider.getLendingPoolConfigurator, we now get a new address. This is the configurator address in the system.



LendingPoolConfigurator: 0xF1bEF1a345371716B607590003f2aBC180C7036C

Following the steps above,

```
LendingPoolParametersProvider: 0x71B78196dFB678A6B796AfaBFe629a1A42BCD32d
```

We'll continue to deploy and initialize all the contracts listed at the top of `lendingPoolAddressProvider`. Note that some of the contracts aren't wrapped in proxies so the address won't change on set.

```

**/

contract LendingPoolAddressesProvider is Ownable {
    //events
    event LendingPoolUpdated(address indexed poolAddress);
    event LendingPoolCoreUpdated(address indexed coreAddress);
    event LendingPoolParametersProviderUpdated(address indexed parametersProviderAddress);
    event LendingPoolManagerUpdated(address indexed managerAddress);
    event LendingPoolConfiguratorUpdated(address indexed configuratorAddress);
    event LendingPoolLiquidationManagerUpdated(address indexed liquidationManagerAddress);
    event LendingPoolDataProviderUpdated(address indexed dataProviderAddress);
    event EthereumAddressUpdated(address indexed ethereumAddress);
    event PriceOracleUpdated(address indexed priceOracleAddress);
    event LendingRateOracleUpdated(address indexed rateOracleAddress);
    event FeeProviderUpdated(address indexed feeProviderAddress);
    event TokenDistributorUpdated(address indexed tokenDistributorAddress);
}

```

```

LendingPool: 0x667091e8c1FA63977C1Ce99Ae5653Af677C17655
LendingPoolCore: 0x6444Bff5143F4722A47877774A39dD9250D529A5
LendingPoolConfigurator: 0xF1bEF1a345371716B607590003f2aBC180C7036C
LendingPoolParametersProvider: 0x71B78196dFB678A6B796AfaBFe629a1A42BCD32d
LendingPoolManager: this is a wallet address (preferably multisig). For
simplicity, I'll initialize it as my kovan deploy address:
0x4011Ba13ACD2E078B0234C125D04e5C61803D07d
LendingPoolLiquidationManager: 0xb0A659f600Ef834BA2c172EA790243bCAbe098ed
LendingPoolDataProvider: 0x9e5d3229359eB3640D60066cBb1de83F216158CE
EthereumAddress - unused in this contract
PriceOracle -> first deployed the provided mock PriceOracle:
0x31DAe81A5c6C435f8d7d43990822775181bA1403
LendingRateOracle: 0xDF9DD372dfd530972e776E91b4C9baD7d334F741
FeeProvider: 0xdd42eb4633aAE0609f72D8163edDa309f2E0f4c6
TokenDistributor: 0x8b48d8eE7B81988df5321f62a73Cb0AaD2B1F412

```

Note on the price oracle: I've given arbitrary values for the ETHUSD and Kovan Dai price so that zero isn't returned. Once the reserves are deployed, I'll update for each token.