Developing a Giveth module for Melon

by Ash

December 2019

TABLE OF CONTENT

1 Introduction

This project is part of the development of Ash, an asset management application running on top of the decentralized fund management protocol Melon. The goal was to enable donations from the Melon protocol i.e. from Melon funds to Giveth DACs (Decentralized Altruistic Communities). One of the main challenges was to connect those two decentralized applications in a shared testing environment, since both systems are located on different testnets (Kovan testnet/Ropsten testnet).

2 Giveth module architecture

To allow testing on the Kovan testnet we deployed a Giveth bridge dummy. There are two main layers in the protocol relevant for our goal:

- solidity files (smart Contracts)
- typescript files (to use the smart contracts)

Solidity

The first layer is the smart contract system setup.

```
function donateOnExchange(
        uint exchangeIndex,
        string methodSignature,
        address bridge,
        uint64 receiverDAC,
        address donationAsset,
        uint donationQuantity
    )
        public
        onlyInitialized
    {
        //registrychecks of Aaptermethod and Asset
        bytes4 methodSelector = bytes4(keccak256(methodSignature));
        require(
            Registry(routes.registry).adapterMethodIsAllowed(
                exchanges[exchangeIndex].adapter,
                methodSelector
            ),
            "adapterMethodIsAllowed failed."
        );
        require(Registry(routes.registry).assetIsRegistered(
            donationAsset), 'donationAsset not registered'
        );
        require(
            exchanges[exchangeIndex].adapter.delegatecall(
                abi.encodeWithSignature(
                    methodSignature,
                    bridge,
                    receiverDAC,
                    donationAsset,
                    donationQuantity
                )
            ),
            "Delegated call to exchange failed"
        );
        donations.push(Donation(exchanges[exchangeIndex], msg.sender, donationAsset, donationQuantity));
    }

```

Here we added in Trading.sol the function donateOnExchange to maintain the syntax and philosophy of callOnExchange. It is not a usual callOnExchange (usually the funds are trading their assets via this function) as we will not receive any assets in return. Hence, we created this new function. Developing a proof of concept, donateOnExchange is restricted to a

givethDonation, and is not implemented as open as callOnExchange. Opening up the code base for other exchangeDonations is a future to-do.

The donateOnExchange function, as well as the callOnExchange setup calls the given ExchangeAdapter, the givethBridgeAdapter.sol especially the makeDonation function which takes a uint receiverDAC, address token and uint amount as inputs. makeDonation then checks if the chosen token can be traded/donated and takes it from the Vault.sol contract of the fund. Hereafter it calls the Giveth bridge contract and executes donateAndCreateGiver. Now the donation has been successful. This exchange adapter can be improved through Giveth by creating a new function in the givethBridge.sol which shows the whitelisted tokens.

```
22      function makeDonation(
23          address bridge,
24          uint64 receiverDAC,
25          address donationAsset,
26          uint donationQuantity
27          ) public onlyManager notShutDown {
28          //prevalidations
29          ensureCanMakeOrder(donationAsset);
30          getTrading().updateAndGetQuantityBeingTraded(donationAsset);
31          ensureNotInOpenMakeOrder(donationAsset);
32          //TODO:  Add in givethBridge (OpenSource Community) 'checkWhitelisted(address token)'.
33          //TODO2: Use it.
34          // Prepare donation
35          prepareDonation(bridge, donationAsset, donationQuantity);
36          //makeDonation
37          bridge.call(
38              abi.encodeWithSignature(
39                  "donateAndCreateGiver(address,uint64,address,uint256)",
40                  msg.sender,
41                  receiverDAC,
42                  donationAsset,
43                  donationQuantity
44              )
45          );
46          // Postprocess/Update
47          getAccounting().updateOwnedAssets();
48          donations[msg.sender][donationAsset] *= donationQuantity;
49          Donated(msg.sender, donationAsset, donationQuantity);
50      }
51
52      function prepareDonation (address bridge, address donationAsset, uint donationQuantity) internal {
53          Hub hub = getHub();
54          Vault vault = Vault(hub.vault());
55          vault.withdraw(donationAsset, donationQuantity);
56          require(
57              ERC20(donationAsset).approve(bridge, donationQuantity),
58              "donationAsset could not be approved");
59      }
```

Typescript

In order to use the smart contracts on Ethereum, a way to talk to the blockchain and get infos, execute functions etc. is necessary. Therefore, we implemented changes and adjustments such as the following:

- adding the exchange adapter and the "exchange" into the deployment process in /src/utils/deploy/, deploySystem.ts and deployThirdParty.ts.
- writing new typescript functions to use the new adapter and prepare the args, set guards etc. (/src/contracts/fund/trading/transactions/, donateOnExchange.ts and donateGiveth.ts)

NOTE: As a proof of concept this module only runs on the Kovan testnet for now. The code is @Midas-Technologies-AG/protocol a fork from @melonproject and especially the setup and following test commands from the Branch kovan-donateOne.

3 Smart contracts and module usage

The path of our added exchange adapter is src/contracts/exchanges/adapters/givethBridgeAdapter.sol. .sol files mostly have a /transactions, or ./calls directory with all the possible, or needed typescript functions the node app can execute.
Finally we added one function in /src/contracts/fund/trading/Trading.sol which uses the adapter givethBridgeAdapter.sol in the path above.
There is one special way, the funds can be taken out of the vault.sol and that is through adapters of the exchanges, which are so called thirdParties. So, the adapter is the accessor to the Melon fund management system. This module of the @melonproject/protocol adds a one-way donation function for funds.
If you have an invested fund you only need to import { donateGiveth } from ... and use it like await donateGiveth(environment, receiverDAC, token, amount) to donate to the receiverDAC (givethDAC on mainnet is “5”), the desired amount of token.

Usage

The fund owner must be able to call tradingAddress.donateOnExchange(...) and give the right inputs:

- exchangeIndex
- bridgeAddress
- receiverDAC
- token
- amount

which then calls givethBridgeAdapter.makeDonation() which then takes amount of token from the vaultAddress.withdraw(), ERC20(token).approve() the bridgeAddress for token and amount, and then calls givethBridge.donateAndCreateGiver() which makes the donation finally.

4 Kovan prototype

Before deploying the Giveth module on the Ethereum mainnet there are a couple of necessary adjustment steps left:

- Further and deeper integration into the policy system etc. of the Melon protocol
- Redeployment of the Giveth bridge with checkWhitelisted(address _token) public view...

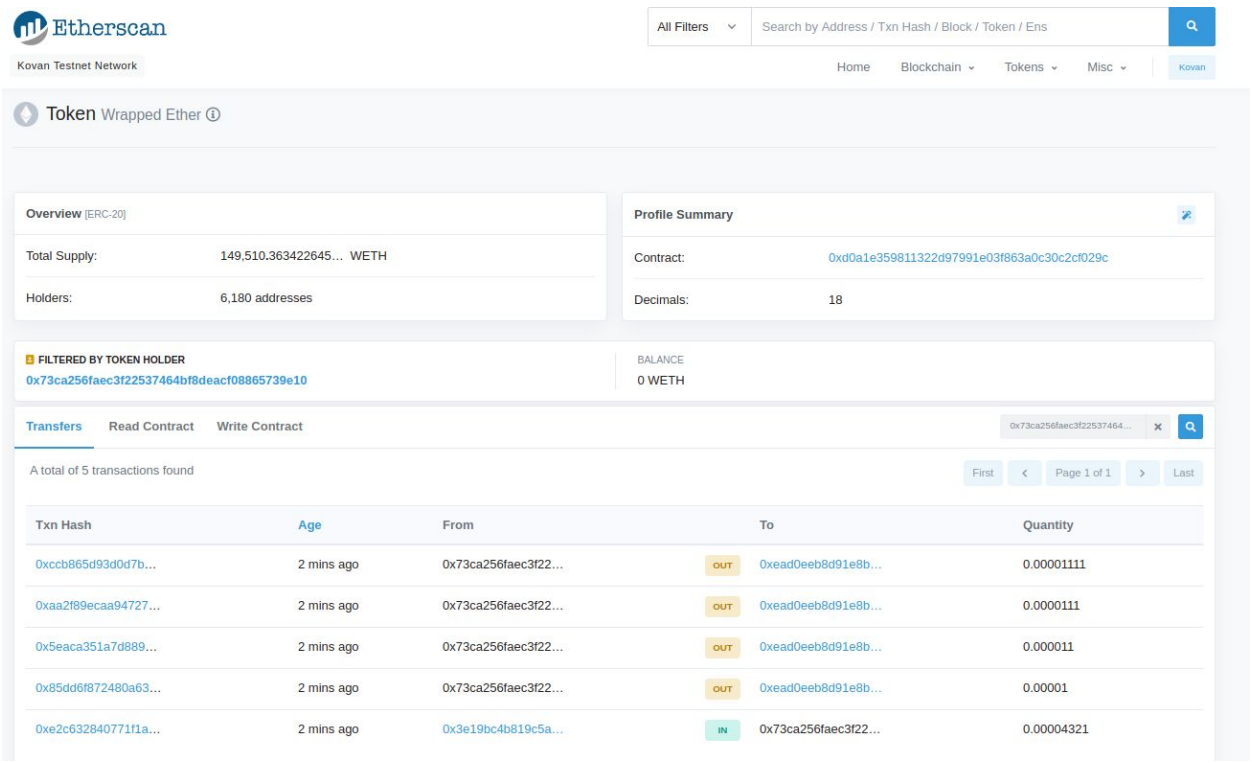
Since the goal is to donate tokens, every token needs to get whitelisted on Giveth in order to be accepted as a donation. We deployed it ourselves to be able to set the whitlistings.

The testing environment is set up the Kovan testnet due to different reasons:

- Usability (for developer)
- the much bigger and complexer solidity system is the protocol and its components as well as the support in the past have been much bigger on the kovan TESTNET chain.

Although it should make no difference to use the mainnet chain, there is a huge difference regarding, "real" costs of transactions, support as well as development hurdles. Redeploying a

system of the size of the Melon protocol from scratch could be dependent on a high gas price variance on Ethereum.
Hence, we deployed everything from scratch on Kovan and you will find the commands with output below.



https://kovan.etherscan.io/address/0x173add8c7e4f7034e9ca41c5d2d8a0a986fd427e

On the picture above and the following link you see eight transactions starting at Block 15,279,692 with tx hash 0x2252970f9…. These are donations from a fund ('Fund1') to the Giveth DAC matching the identifier receiverDAC. With the setting from the following commit on the Midas Github, namely the branch kovan-donateOnE we run the two combined commands below to setup this testSetup from scratch with giving two export variables: PRIVATE_KEY=... and JSCON_RPC_ENDPOINT=wss://infura....
The created fund has following routes:

```
2019-12-06T16:07:16.766Z info Midas-Technologies-AG/protocol:givethTests:: "setup Fund was successfull" {
  "accountingAddress": "0xDCbE3a3698F634f2A79D74e50B03C745054c8770",
  "feeManagerAddress": "0x532AC11b8B1A543f46F3FB0a24E8e41D40df2478",
  "participationAddress": "0x3E19Bc4b819C5A7CD6B3F60F78fB2B0A8AA7fe26",
  "policyManagerAddress": "0x735172ADa7876635f75b62bcEa79DfF1870ACE29",
  "priceSourceAddress": "0x385a59e848f6456ADf19C367c8cf03FD39c23FAB",
  "registryAddress": "0x563FeD76573b6f5240b6E136673f9D1179C6c48E",
  "sharesAddress": "0xfc967d9CAe96D3F665D2D103f1e267f4839135e9",
  "tradingAddress": "0xEAD0EeB8d91e8BeE56720666688e802c1c888c7A",
  "vaultAddress": "0x73CA256fAEc3F22537464bF8deAcF08865739E10",
  "versionAddress": "0xE88A7ECe8F8B0E8CA5bC3A33675541f2eA76D180",
  "hubAddress": "0x33350cF81F76d4656FE35e4A756797C6Bcc4CE70"
}
```

IMPORTANT: If you want to redo, set the variables above via bash export PRIV.... and change in the bin/*deploy*.sh file which gets called via yarn dplK via package.json command by removing the line --keystore and adding -P <your-private-key> \ or your personal keystore file.