Multisig deployment

warning This page is massively outdated with the latestLido V2 release . This HOWTO describes deployment of the DAO using a multisig/airgapped signer, step-by-step.

Preparation

Clone the repo and install the deps:

git clone git@github.com:lidofinance/lido-dao.git cd lido-dao yarn Running deployment scripts requires RPC connection to an Ethereum client, which can be configured by editing thehardhat.config.js file. It is already pre-configured for using the Infura provider, just copyaccounts.sample.json toaccounts.json and edit theinfura key:

{ "eth": {}, "infura": { "projectId": "PUT_YOUR_PROJECT_ID_HERE" } } Some of the deployment steps (namely, deploying contracts) cannot be performed from some multisig providers and thus require sending the transactions from a usual address. The repo provides a helper for doing that; if you plan to use it, editaccounts.json and put your accounts config under theeth. key. If your RPC client provides an unlocked account, useremote as the value (here and later we assume that the target network is namedmainnet):

{ "eth": { "mainnet": "remote" }, "infura": { "projectId": "PUT_YOUR_PROJECT_ID_HERE" } } If you plan to use a BIP-44 mnemonic phrase instead, use the following config shape:

{ "eth": { "mainnet": { "mnemonic": "YOUR_MNEMONIC_HERE", "path": "m/44'/60'/0'/0", "initialIndex": 0, "count": 1 } }, "infura": { "projectId": "PUT_YOUR_PROJECT_ID_HERE" } } You can test the config correctness by listing the accounts and their balances:

yarn hardhat --network mainnet list-accts

Deployment steps

The deployment process consists of multiple steps. Generally, after each step a set of transaction files is generated. These transactions need to be executed in a sequential order: only send the next transaction after the previous one is included in a block. After the last transaction from a certain step is included in a block, move to the next step.

There's also a couple steps that don't generate any transactions but check the correctness of the previous steps instead.

1. Deploying the base implementations and the template

Lido uses upgradeable proxy contracts as storage for the state. Each proxy contract points to an implementation contract providing the code that reads and mutates the state of the proxy. Implementation contracts can be upgraded via DAO voting. Implementations are immutable, they are only allowed to modify the caller's (i.e. proxy) contract state.

In order to setup the protocol, one needs to deploy initial versions of the implementations. Some popular multisig vaults, e.g. Gnosis Safe, don't support deploying new contracts so this has to be done from a usual address.

Part of the protocol deployment logic is incorporated in a contract calledLidoTemplate.sol , which also needs to be deployed prior to running further steps.

Prepare the network state file

The deployment scripts use a JSON file nameddeployed-.json to read the initial environment and protocol configuration and to store data that needs to be persisted between deployment steps. If a deployment step requires anything except RPC endpoint and ETH accounts, then it needs to be specified in the network state file. These files are meant to be added under the source control. If some data is missing from the file, the deployment step will fail with an error saying what's exactly missing.

The first step requires the following values:

- networkId
- id of the network
- ensAddress
- ENS registry address
- daoFactoryAddress
- AragonDAOFactory
- contract address
- apmRegistryFactoryAddress
- AragonAPMRegistryFactory

- address
- miniMeTokenFactoryAddress
- AragonMiniMeTokenFactory
- address
- aragonIDAddress
- aragonIDFIFSResolvingRegistrar
- address
- multisigAddress
- the address of the multisig contract that will be used in the next steps
- to perform the further deployment

For example, a network state file formainnet will be nameddeployed-mainnet.json and will initially look like this:

{ "networkId": 1, "ensAddress": "0x00000000000C2E074eC69A0dFb2997BA6C7d2e1e", "daoFactoryAddress": "0x7378ad1ba8f3c8e64bbb2a04473edd35846360f1", "apmRegistryFactoryAddress": "0xa0BC4B67F5FacDE4E50EAFF48691Cfc43F4E280A", "miniMeTokenFactoryAddress": "0x909d05f384d0663ed4be59863815ab43b4f347ec", "aragonIDAddress": "0x546aa2eae2514494eeadb7bbb35243348983c59d", "multisigAddress": "YOUR_MULTISIG_CONTRACT_ADDRESS" } Please note that settingmultisigAddress correctly is very important: this address will own the deployed template contract, and so only this address will be able to perform the deployment steps starting from Lido APM deploy (step 5).

Generate transaction data files

After preparing the values in network state file, generate a set of JSON files with transaction data:

yarn hardhat --network mainnet run ./scripts/multisig/01-deploy-lido-template-and-bases.js

Network ID: 1 Reading network state from /Users/me/lido-dao/deployed-mainnet.json... ==================== Saving deploy TX data for LidoTemplate to tx-01-1-deploy-template.json Saving deploy TX data for Lido to tx-01-2-deploy-lido-base.json Saving deploy TX data for LidoOracle to tx-01-3-deploy-oracle-base.json Saving deploy TX data for NodeOperatorsRegistry to tx-01-4-deploy-nops-base.json ==================== Before continuing the deployment, please send all contract creation transactions that you can find in the files listed above. You may use a multisig address if it supports deploying new contract instances. ==================== Writing network state to /Users/me/lido-dao/deployed-mainnet.json... All done!

Send the transactions

You can use thetx helper for sending the transactions from files. It supports the following flags:

- --from
- the sender address
- --file
- the TX file which may contain the following fields:to
- ,value
- ,data
- ,gas
- ,from
- --gas-price
- gas price in wei (optional)
- --nonce
- sender nonce (optional)
- --wait
- the number of seconds to wait before sending the tx (optional, default 5)

Run the following to deploy the implementations and the template:

yarn hardhat --network mainnet tx --from DEPLOYER --file tx-01-1-deploy-template.json yarn hardhat --network mainnet tx --from DEPLOYER --file tx-01-2-deploy-lido-base.json yarn hardhat --network mainnet tx --from DEPLOYER --file tx-01-3-deploy-oracle-base.json yarn hardhat --network mainnet tx --from DEPLOYER --file tx-01-4-deploy-nops-base.json You're not required to use this helper to send the transactions defined in the generated files; it's there for the convenience only.

This step is an exception from the "sequential transactions" rule: you can send all four transactions in parallel from different addresses.

Update the network state file

After all four transactions are included in the blockchain, update the network state file with the following values:

- daoTemplateDeployTx
- hash of the TX sent from thetx-01-1-deploy-template.json
- file
- lidoBaseDeployTx
- hash of the TX sent from thetx-01-2-deploy-lido-base.json
- file
- oracleBaseDeployTx
- hash of the TX sent from thetx-01-3-deploy-oracle-base.json
- file
- nodeOperatorsRegistryBaseDeployTx
- hash of the TX sent from thetx-01-4-deploy-nops-base.json
- file

2. Verifying the deployed contracts

Run the following:

yarn hardhat --network mainnet run ./scripts/multisig/02-obtain-deployed-instances.js This step will verify the deployed contracts and add the following fields to the network state file:

- daoTemplateAddress
- address of theLidoTemplate
- contract
- app:lido.baseAddress
- address of theLido
- implementation contract
- app:oracle.baseAddress
- address of theLidoOracle
- implementation contract
- app:node-operators-registry.baseAddress
- address of theNodeOperatorsRegistry
- implementation contract

3. Register a ENS domain for Lido APM

This ENS domain is needed for Aragon Package Manager (APM) instance that the protocol will use for the upgrade mechanics. Prior to running the step, add the following keys to the network state file:

- lidoApmEnsName
- the second-level ENS domain that APM will use to register packages
- lidoApmEnsRegDurationSec
- the domain lease duration in seconds

Then, run:

yarn hardhat --network mainnet run ./scripts/multisig/03-register-ens-domain.js ... ==================== Saving data for commit transaction to tx-02-1-commit-ens-registration.json (projected gas usage is 53667) Saving data for register transaction to tx-02-2-make-ens-registration.json ==================== Before continuing the deployment, please send all transactions listed above.

Make sure to send the second transaction at least 60 seconds after the first one is included in a block, but no more than 86400 seconds after that. ==================== The step will generate two transaction files. You'll need to send these transactions one after another, waiting no less than one minute between them:

yarn hardhat --network mainnet tx --from DEPLOYER --file tx-02-1-commit-ens-registration.json sleep 60 yarn hardhat --network mainnet tx --from DEPLOYER --file tx-02-2-make-ens-registration.json

4. Deploy Lido frontend apps

The Lido DAO includes frontend apps for DAO governance and protocol management. They are deployed to IPFS, so you'll need to specifyipfsAPI key in the network state file pointing to an IPFS client API endpoint, e.g."ipfsAPI": "http://localhost:5001/api/v0" . Then, run the following:

yarn hardhat --network mainnet run ./scripts/multisig/04-publish-app-frontends.js Make sure that either the IPFS node you're using is going to be permanently up and publicly available, or that you pin the uploaded content to some other permanent public node.

This step will addipfsCid andcontentURI subkeys for all three Lido apps (app:lido ,app:oracle ,app:node-operators-registry) in the network state file. The first key is the IPFS identifier for the root entry of the app frontend, andcontentURI is the same key encoded to an Aragon-specific format.

5. Deploy Lido APM

Run the following:

yarn hardhat --network mainnet run ./scripts/multisig/05-deploy-apm.js ... ==================== Parent domain: eth 0x93cdeb708b7545dc668eb9280176169d1c33cfd8ed6f04690a0bcc88a93fc4ae Subdomain label: lidopm-pre 0x1353eb779a45ed66bdb49e45e006df81a69d9f73067e846003b5bb00984191d4 ==================== Saving data for APM deploy transaction to tx-03-deploy-apm.json (projected gas usage is 6263517) ==================== The step will generate a transaction file; you'll need to send this transaction from the contract atmultisigAddress . After the transaction is included in a block, move to the next step.

Using Gnosis Safe

If you're using Gnosis Safe, this can be done by choosingNew Transaction > Contract Interaction and enabling theUse custom data (hex encoded) option in the popped dialog. Then, copy the contents of theto key from the transaction JSON file to theRecipient *field, the contents of thevalue field to theValue*field (enter0 if there's novalue key in the transaction JSON), and the contents of thedata field to theData (hex encoded)* field.

Make sure to check the gas limit of the transaction: Gnosis Safe frequently sets it too low. As a rule of thumb, set it to the value of thegas key in the transaction JSON file plus1500000 (the additional gas is used to handle multisig logic).

6. Check the deployed APM

Run the following:

yarn hardhat --network mainnet run ./scripts/multisig/06-obtain-deployed-apm.js Make sure that it finishes without errors and move to the next step. The following field will be added to the network state file:

- lidoApmAddress
- the address of the Lido APM controllinglidoApmEnsName
- ENS domain.

7. Create application APM repositories

Run the following:

yarn hardhat --network mainnet run ./scripts/multisig/07-create-app-repos.js ... ==================== Saving data for createRepos transaction to tx-04-create-app-repos.json (projected gas usage is 7160587) ==================== The step will generate a transaction file; you'll need to send this transaction from the contract atmultisigAddress . After the transaction is included in a block, move to the next step.

8. Deploy DAO and its governance token

This step will deploy the instances of the DAO and governance token. You'll need to add a field calleddaoInitialSettings to the network state file prior to running the step:

// ... "daoInitialSettings" :

{ // Governance token name/symbol; cannot be changed post-deploy "token" :

{ "name" :

"Lido DAO Token" , "symbol" :

"LDO" } , // Beacon chain spec; can be changed via DAO voting "beaconSpec" :

{ "depositContractAddress" :

"0x00000000219ab540356cBB839Cbe05303d7705Fa" , "slotsPerEpoch" :

32 , "secondsPerSlot" :

12 , "genesisTime" :

1606824023 , "epochsPerFrame" :

225

// Lido oracles report once per epochsPerFrame epochs } , // DAO voting configuration (Aragon Voting app) "voting" :

{ "minSupportRequired" :

"500000000000000000" ,

// 1e18 === 100% "minAcceptanceQuorum" :

"50000000000000000" ,

// 1e18 === 100% "voteDuration" :

172800

// in seconds } , // Protocol fee configuration; can be changed via DAO voting "fee" :

{ "totalPercent" :

10 , "treasuryPercent" :

0 , "insurancePercent" :

50 , "nodeOperatorsPercent" :

50 } } // ... Then, run the following:

yarn hardhat --network mainnet run ./scripts/multisig/08-deploy-dao.js ... Saving data for newDAO transaction to tx-05-deploy-dao.json (projected gas usage is 7118882) Send the generated transaction from the contract atmultisigAddress . After the transaction is included in a block, move to the next step.

9. Check the deployed DAO

Run the following:

yarn hardhat --network mainnet run ./scripts/multisig/09-obtain-deployed-dao.js Make sure that it finishes without errors and move to the next step. The following fields will be added to the network state file:

- daoAddress
- the address of the DAO instance;
- daoTokenAddress
- the address of the DAO governance token;
- proxyAddress
- keys underapp:*
- keys: addresses of the app instances.

10. Issue DAO governance tokens

Add thevestingParams key to the network state file containing the following:

// ... "vestingParams" :

{ // unvested tokens will be held on the DAO Agent app "unvestedTokensAmount" :

"10000000000000000000000" , // token holder addresses and their respective amounts "holders" :

{ "0xaabbcc0000000000000000000000000000000000" :

"100000000000000000000" , // ... } , // Vesting start date "start" :

1608213253 , // Vesting cliff date "cliff" :

1608213253 , // Vesting end date "end" :

1608501253 , // Whether vestings should be revokable by the DAO "revokable" :

false // See https://github.com/aragon/aragon-apps/blob/master/apps/token-manager/contracts/TokenManager.sol } // ... Then, run the following:

```
yarn hardhat --network mainnet run ./scripts/multisig/10-issue-tokens.js ... ==================== Total batches: 2 Saving data for issueTokens (batch 1) transaction to tx-06-1-issue-tokens.json (projected gas usage is 6478755) Saving data for issueTokens (batch 2) transaction to tx-06-2-issue-tokens.json
```

Send the generated transactions sequentially from the contract atmultisigAddress , waiting until the first one is included in a block before sending the second one. After the second transaction is included in a block, move to the next step.

11. Finalize the DAO

Add thedaoAragonId key to the network state file, setting it to a name that the DAO will be registered by in aragonID, i.e..aragonid.eth will resolve to thedaoAddress . Run the following:

```
yarn hardhat --network mainnet run ./scripts/multisig/11-finalize-dao.js ... ==================== Saving data for finalizeDAO transaction to tx-07-finalize-dao.json (projected gas usage is 5011582)
```

Send the generated transaction from the contract atmultisigAddress . After the transaction is included in a block, move to the next step.

12. Perform the final checks

At this point, the DAO is fully deployed. Run the following to verify the correctness of the configuration and permissions setup:

```
yarn hardhat --network mainnet run ./scripts/multisig/12-check-dao.js
```

If there's some error, it will be printed and further checks will be cancelled. This step only requires the following fields to be defined in the network state file:

- ensAddress
- lidoApmEnsName
- daoAragonId
- vestingParams
- daoInitialSettings
- daoTemplateAddress Edit this page Previous Keys API Next Protocol levers