Threshold Docs

What is the Threshold Network?

Getting Started

Got 2 minutes? Check out an introduction video to the Threshold Network:

Staking Guides: Jump right in

Follow our handy guides to get started on staking basics as quickly as possible:

Staking on Threshold

Fundamentals: Dive a little deeper

Learn the fundamentals of the Threshold Network to get a deeper understanding of our main features:

Proxy Re-Encryption (PRE)

tBTC v2

Threshold DAO

Fundamentals

Threshold Access Control

Threshold Application

Threshold Access Control enables end-to-end encrypted data sharing and communication without the need to trust a centralized authority. It is the only access control layer available to Web3 developers that has **already achieved true decentralization** through a live and well-collateralized network, with sensitive cryptographic operations disassembled and distributable across ~250 independently operated servers.

End-users of applications which have integrated Threshold Access Control enjoy the following features & benefits:

- *End-to-end encryption for everything*.
  Built on the privacy-for-everyone principles of popular end-to-end encrypted messengers but applicable to a far wider set of use cases, including private NFTs, connected vehicles, DBaaS, live-streaming, private DAO group chats, and much more.
- *Trustlessness via true decentralization.*
  Key management, condition verification and ciphertext re-encryption are operationally distributed across a geographically diverse array of machines/servers, operated by economically independent individuals and commercial entities.
- *Powerful, per-ciphertext conditionality.*
  Future access to data can be contingent on the fulfillment of nearly any predefined condition, and those conditions attached to any granularity of data payload (e.g. a single message or an entire table).
- *Flexible condition composability.*
  Conditions of all types can be mixed-and-matched using logical operators and flexible prefix notation into virtually any desired combination.
- *Tunable collusion-resistance, redundancy & latency.*
  Developers have full control over the cohort(s) of node operators which manage access to a given data payload, user base or entire application. These security parameters can also be packaged into simpler user-facing optionality for custom risk preferences.
- *Highly incentivized uptime.*
  The Threshold network's multi-app model strongly incentivizes node operators to provision service to tBTCv2, and its strict availability requirements. Threshold Access Control 'piggybacks' on tBTCv2 uptime, reliability and technical competence.
- *Optional: keypair-only decryption.*
  If even stricter security guarantees are required, and data recipients' public keys are known in advance, developers may opt for end-user data to be re-encrypted by node operators such that they are only decryptable by pre-designated clients.

The Threshold Access Control service is built on two distinct but interwoven technologies; *Conditions-Based Decryption* (CBD) and *Proxy Re-Encryption* (PRE). Both offer trust-minimized end-to-end encryption and access control is executed by the same decentralized array of nodes. There are some trade-offs between (explicit) security and access condition customization, which are explored in the trust assumptions section.

Note for Threshold stakers: the up-and-running 'PRE app' will eventually be renamed to encompass the broader Threshold Access Control service, and will also acquire new functionality via the addition of CBD technology. This will require a DAO-driven upgrade but will not fundamentally change the operational requirements or compensation.

Conditions-Based Decryption (CBD)

Threshold Application – underlying technology

Conditions-Based Decryption (CBD) is one of two technologies underpinning the Threshold Access Control service, the other being Proxy Re-Encryption. The vast majority of developers will only require CBD technology to provably protect their users' data, and we recommend that all developers start here – by familiarizing themselves with CBD.

In broad terms, CBD enables the users of adopting applications to specify conditions for accessing their encrypted data. If those conditions are not fulfilled – which means formal verification by a cohort of nodes from the Threshold network – then the data remain entirely unreadable to anyone besides the original data owner/encryptor. However, if the predefined conditions are provably satisfied, then the data requester gains decryption rights.

Threshold Decryption

Under the hood, CBD involves splitting a joint secret – a decryption key – into multiples *shares* and distributing those among authorized and collateralized node operators (stakers in the Threshold network). A minimum number – a *threshold* – of those operators holding the key shares must be online and actively participate in partial decryptions. These are subsequently combined on the requester's client to reconstruct the original plaintext data.

Conditionality

A range of access condition types can be defined by the data owner. For example:

- EVM-based
  e.g. Does the requester own a given NFT?
- RPC-driven
  e.g. Does the requester have at least X amount of a given token in their wallet?
- Time-based
  e.g. Has a predefined period elapsed, after which requests will be ignored?

These conditions are also composable and can be combined in any logical sequence or decision tree.

Conditions are 'attached' on a per-ciphertext basis. In other words, each and every payload, message or bit can be access-restricted by a unique set of specified conditions. In most situations, condition sets will be automatically reused until the end-user proactively configures them – for example, in order to remove an address from continuing to access the messages in a group chat.

Conditions Verification

Currently, requesters prove their association with condition fulfillment – i.e. their right to receive a threshold number of decrypting shares – by signing a transaction that verifies their ownership of a given Ethereum wallet. That wallet is checked for fulfillment of the specific condition – e.g. owning an NFT in order to access a DAO's knowledge base.

This signature can be cached by the application for a use case-appropriate period of time, such that the user does not have to repeatedly re-sign for access later. However, verification still takes place in the background – for example, if a wallet address is removed from an NFT-gated group-chat by dint of changing the access-granting NFT, they will immediately be unable to see new messages. The cached signature will not give them access any longer than is specified by the data owner/encryptor.

Network Parameterization

CBD is fully decentralized. Therefore, developers have the option to tweak certain network parameters, which affect the collusion-resistance, redundancy, latency and (in future versions) cost of using CBD. For example:

- The number of decrypting shares *n*
- The frequency and/or prompt for resampling a new cohort from the node array
- The 'hard-coded' node address that always feature in the cohort

This optionality can also be surfaced for end-users – for example, in the form of 'packages' that they might choose between, accommodating their risk and cost preferences . A guide on best practices for choosing/surfacing network parameters will be released soon.

Proxy Re-Encryption (PRE)

Threshold application – underlying technology

PRE is an end-to-end encryption protocol that is a more scalable, more flexible form of public-key encryption and enables a group of proxy entities to transform encrypted data from one public key to another, without the power to decrypt the data or gain access to any private keys. PRE equips developers, applications and end-users with **secrets management** and **dynamic access control** capabilities. The nodes on the Threshold Network act as these proxy entities and use threshold cryptography to securely and cooperatively re-encrypt data for recipients based on access conditions defined by the data owner.

PRE is directly applicable to use cases that aim to maintain data ownership while facilitating data sharing capabilities such as paid subscriptions to encrypted content, or the transfer of data ownership for encrypted NFTs. Data, wherever stored, remains private and encrypted while data owners maintain the ability to share that data and cryptographically enforce access controls.

Want to get started staking with Threshold and running a node for PRE? Follow this guide:

PRE Node Setup

tBTC v2

Threshold application

Existing solutions that bridge Bitcoin to Ethereum require users to send their Bitcoin to an intermediary, in exchange for an Ethereum token that represents the original asset. This centralized model requires you to trust a third party and is prone to censorship, threatening Bitcoin's promise of secure, permissionless decentralization.

The second generation of tBTC (tBTC v2) is a truly decentralized bridge between Bitcoin and Ethereum. It provides Bitcoin holders secure and open access to the broader cryptoeconomy. tBTC v2 allows you to unlock your Bitcoin’s value to borrow and lend, mint stablecoins, provide liquidity, and much more.

Instead of centralized intermediaries, tBTC v2 uses a randomly selected group of operators running nodes on the Threshold Network to secure deposited Bitcoin through threshold cryptography. That means tBTC v2 requires a threshold majority agreement before operators perform any action with your Bitcoin. By rotating the selection of operators weekly, tBTC v2 protects against any individual or group of operators seizing control. Unlike other solutions on the market, users of tBTC v2 trust math, not hardware or people.

tBTC

Threshold USD

Threshold application

Threshold USD (thUSD) is a tBTC backed stablecoin pegged 1:1 against USD.

Background: About Bootstrapping

A certain network effect is required for a stablecoin to function well. Bootstrapping is the process of ensuring the protocol has gained enough traction to become self-sufficient.

In the past this was usually accomplished by being an early adopter of coins, such as buying Bitcoin early and benefiting from the lower price. DeFi revolutionized the bootstrapping concept by enabling the bootstrapping of capital through yield-farming (aka rented liquidity).

Liquity Protocol bootstrapped by issuing LQTY tokens to depositors in the stability pool. This incentivized people to take out loans and deposit to the stability pool.

For normal, healthy operations of the protocol, it's vital that there is sufficient funds in the stability pool to cover all liquidations. However the LQTY incentive resulted in far more deposits than is required and this is essentially wasted capital. Furthermore there are concerns on how sustainable Liquity will be after the initial 100 million LQTY has been issued and the pool is only sustained through liquidations rewards.

How does Threshold USD approach this?

In Threshold USD we take a different approach. A newer concept in DeFi known as Protocol Controlled Value (PCV) is the idea that the protocol itself can own liquidity and use that to improve the protocol. This is a new alternative to renting it elsewhere (aka yield-farming) and has the benefit of long-term sustainability.

But the PCV has to come from somewhere. In thUSD we resolve this by issuing an Initial Protocol Loan to the PCV which then deposit these funds directly to the stability pool. Read more about this here:

Initial Protocol Loan

That lets us bootstrap the stability pool at zero cost to the protocol.

But not only that, by eliminating the LQTY token, **all profits** (from interest, etc) will go directly into the PCV. Compared to Liquity, where profits are distributed among LQTY holders, we have eliminated a massive drain on the system, at no cost.

Having the stability pool funded on its own is enough for the protocol to function, and we can expect some users that want to borrow against their tBTC to take part, but it's not going to create major adoption, and for thUSD to function well it needs to be stable at ~$1, that requires people to put up liquidity at decentralized exchanges.

In order to draw in more users and create liquidity for thUSD, we will issue rewards to pools on other platforms (such as a thUSD stablecoin pool on curve). This will incentivize users to take debt and deposit into curve, with the added benefit of improved liquidity and resiliency.

How fast we want to grow can be almost entirely adjusted by how much reward is sent to the pool.

This part is indeed rented liquidity with all its drawbacks, but remember that Threshold USD profits are sent to the PCV instead of LQTY holders: As the protocol grows in popularity, profits from the protocol itself can be used to fund these rewards, thus creating a self-sustaining feedback loop.

On top of that, Threshold DAO will store some of its reserves in stablecoins, so the Threshold DAO can initiate a PCV with deposits on curve. That will result in more stability for thUSD which is important for adoption.

Initial Protocol Loan

The Initial Protocol Loan (IPL) is a debt issued against the protocol itself in order to fund the stability pool. Even though it's technically debt, it does not in itself make the protocol a fractional reserve because all funds are accounted for.

How it works

thUSD is minted by the PCV and deposited into the stability pool. A freeze on withdraw is initiated until debt is fully repaid. At any time, the Threshold DAO can initiate a withdraw, which will destruct the debt and withdraw exceeds. If the PCV for some reason has less than the debt available, withdraw is denied.

Profits from usage of the protocol (loan interest, redemption fee etc) accrues into the PCV, so in an event where there are less funds than debt the protocol first has to earn back the missing funds.

Benefits of the hybrid PCV + IPL model:

- Bootstrap stability pool for free
- No need for LQTY token = all profits accrue directly to the PCV
- Immediately surplus on first loan drawn / liquidation
- Predictability and (once debt is repaid) high resilience against Black Swans
- No idle capital (it doesn't "really" exist)

Disadvantages

- Higher risk of undercollateralization during the initial stages
- Poor management of PCV could result in not enough funds in the stabilty pool

About the risk

Each liquidation that occurs will draw against the stability pool and result in ~10% profit to the stability pool. This also means that the price can drop up to 10% after a liquidation before the stability pool is at risk of losing money on the liquidation. Our integration with B.Protocol ensures that liquidated tBTC is automatically converted back to thUSD and re-deposited into the stability pool. Under normal circumstances and without other factors, it is expected that the balance of thUSD in the pool will grow over time.

But there can be black swan events, large drops in the price of BTC in a short period of time or liquidity issues that results in a loss, even at ~10% profit. This is a risk of the protocol becoming undercollateralized, but the same issue would apply even without debt (stability pool not being profitable). The protocol is created to be sustainable, so these types of rare events will be averaged out. In addition, all income streams from Threshold USD also ends up in the PCV which further decrease the risk of undercollaterization.

The undercollaterialization is mostly a concern during the initial bootstrapping phase. As interest accrue and repay the initial debt, the funds in the stability pool will be replaced by a fully owned PVC, thus eliminating this disadvantage.

B. Protocol

B. Protocol is a third-party decentralized backstop liquidity protocol aiming to make lending platforms more stable.

The problem

In Liquity Protocol there's a concept called "Stability Pool" [2]. The purpose of the stability pool is to purchase liquidated collateral (tBTC) at a discount by using Threshold USD (thUSD) as payment.

Users deposits thUSD into the stability pool and their funds are pooled with other depositors. If a pool consist of 900,000 thUSD and a user deposits 100,000 thUSD, that user is entitled to 10% of the collateral seized.

The issue with stability pool is that as liquidations occurs, thUSD is traded to tBTC, but never back to thUSD. If left untouched, the pool will eventually only consist of tBTC and no further liquidations can take place against the pool. Furthermore, by acquiring tBTC collateral, the value for depositors in the pool becomes subject to the price of BTC.

Users are therefore incentivized to quickly sell of the tBTC for thUSD to avoid taking on price risk, but in Liquity, this process is manual. User has to manually open the UI and withdraw tBTC, sell it for thUSD and then re-deposit the thUSD. This is both a time consuming and gas costly operation, best suited for whales and users with their own bots.

Even worse, a DAO cannot realistically operate in the current stability pool model because voting to move and sell funds becomes impractical, costly and slow.

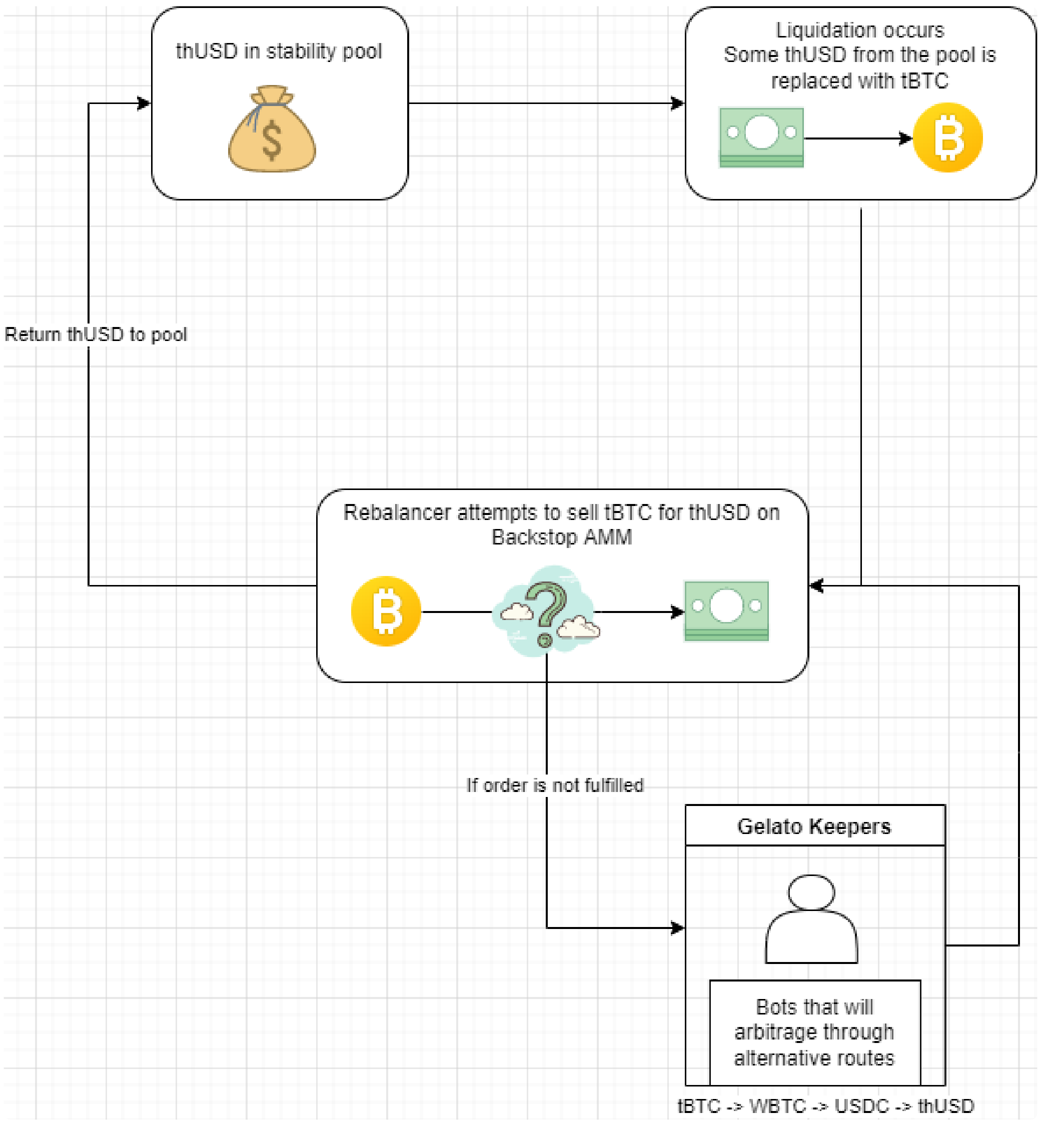
In order to establish a non-human, algorithmic, fully automated solution to compound profits we look to B.Protocol.

B.Protocol to the rescue

Instead of depositing directly into the stability pool, we can use B.Protocol's wrapping smart contract interface, which deposits the thUSD to the stability pool on behalf of all users (and the PCV).

Once liquidation happens, the discounted tBTC is automatically offered for sale by B.Protocol’s Backstop AMM (B.AMM). This is done according to a deterministic formula, which takes into account the current tBTC and thUSD inventory, and the current BTC-USD market price (which is taken from Chainlink). Whenever the sale occurs, the smart contract deposits the returned thUSD back to the stability pool.

If there are no takers for the offer on the B.AMM, Gelato Keepers will arbitrage through popular DEX pairs to fulfill the order.


thUSD in stability pool
Liquidation occurs
Some thUSD from the pool is replaced with tBTC
Return thUSD to pool
Rebalancer attempts to sell tBTC for thUSD on Backstop AMM
If order is not fulfilled
Gelato Keepers
Bots that will arbitrage through alternative routes
tBTC -> WBTC -> USDC -> thUSD

This way the PCV becomes self-sufficient and able to operate indefinitely without outside interference.

Integration details

B.Protocol Team will deploy a non-upgradable pool that is Threshold USD compatible.

The pool will be using Chainlink BTC/USD oracle (same as Threshold USD)

The pool will not deploy idle funds in any yield farming applications.

Threshold USD PCV will whitelist the smart contract for the B.Protocol pool and the Threshold Network DAO (T DAO) will vote to initiate a deposit of thUSD to the B.Protocol pool. The deposit is moved from the PCV smart contract to B.Protocol and can be moved back to the PCV at any time, without limitation, as long as the T DAO votes to do so. If there is tBTC in the pool at the time of withdraw, the tBTC may also be transferred back to the PCV.

The B.AMM contract's "A" parameter ("A" is amplification factor [1], higher value means less slippage) is operated through a co-ownership between T DAO and B.Protocol, where B.Protocol propose "A" value and T DAO can approve or reject.

The discount rate is set to 4% on deployment, but the max rate should be configurable up to 10% in a joint governance mechanism between T DAO & B.Protocol.

All profits from liquidations will auto-compound within the B. Protocol pool.

Keeper

As a backup mechanism, B. Protocol will deploy Gelato keepers [3] that can route.

Example:

```
tBTC -> WBTC -> USDC -> thUSD
```

In this case funds are routed through the following Curve Finance stable pools:

In addition, funds are routed through uniswap (v2 or v3) WBTC/USDC pools.

Other routes could be:

```
tBTC -> WBTC -> DAI -> thUSD
tBTC -> WBTC -> USDT -> thUSD
```

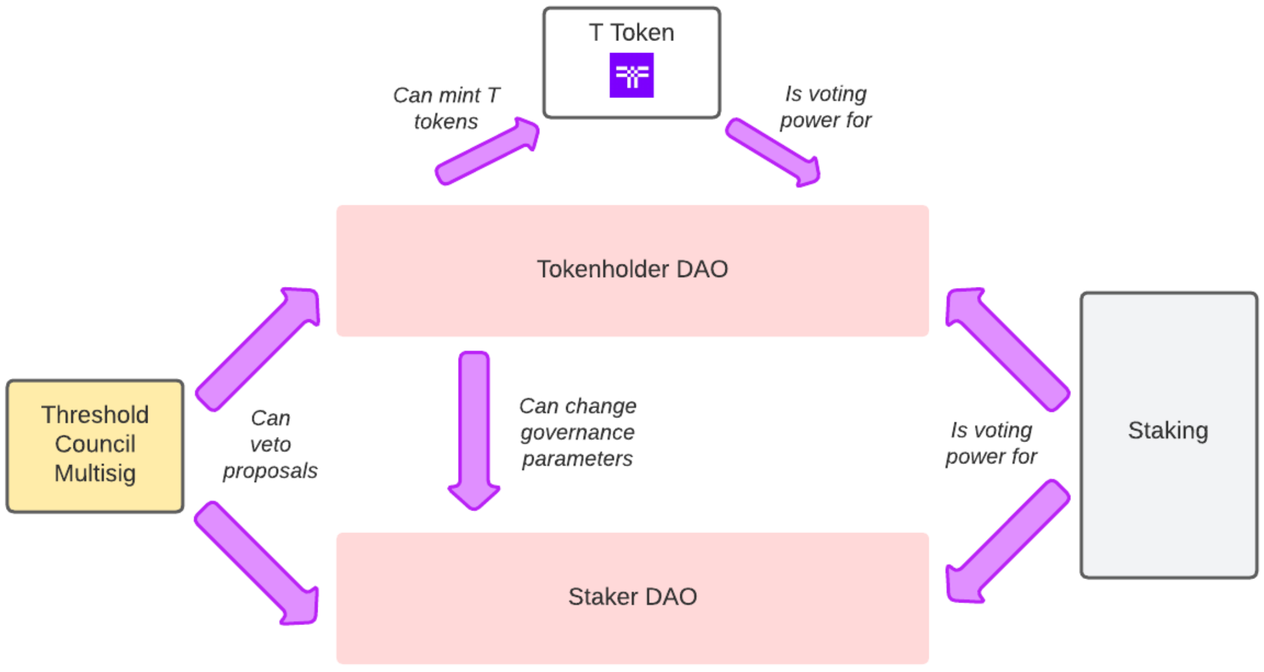
or:

```
tBTC -> renBTC -> USDC -> thUSD
```

The keeper is funded by the DAO and any profit it makes is distributed back to the PCV.

Threshold DAO

Threshold is community-driven and governed by an eponymous DAO that includes the constituencies of both the NuCypher and Keep networks. The community decided that a . The Threshold DAO has three primary bodies: **Tokenholder DAO**, **Staker DAO**, and the **Elected Council**. The goal of this three-pronged approach is to enhance representation while ensuring accountability, as each of these governance bodices will hold the other two accountable, similar to the system of checks and balances found in most constitutional governments. They will also hold separate responsibilities that are embedded in the governance structure. See this blog post and this proposal for a more extended description of the DAO structure.



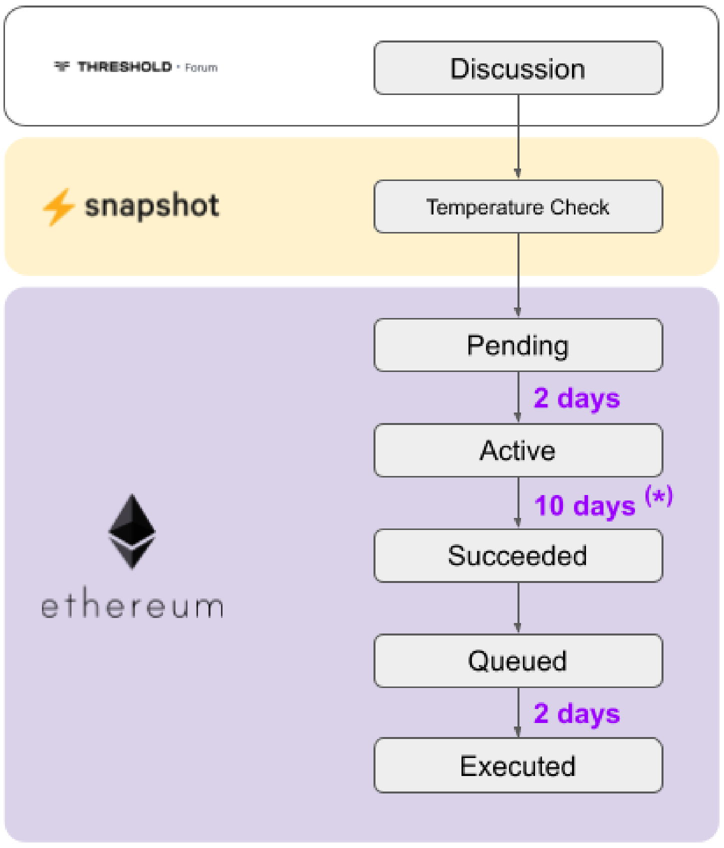
From an implementation perspective, both the Tokenholder DAO and the Staker DAO are based on Governor Bravo governance model (in particular, using OpenZeppelin Governance). The Tokenholder DAO is on Ethereum Mainnet at `0xd101f2B25bCBF992BdF55dB67c104FE7646F5447`. The Elected Council is implemented as a Gnosis Safe contract, with a 6-of-9 configuration, also deployed on Ethereum Mainnet, at `0x9F6e831c8F8939DC0C830C6e492e7cEf4f9C2F5f`. The Staker DAO is postponed until the complete functionality for staking is released.

🗳️ Governance Process

During the early days of the merger, one of the issues discussed by the community was the design of the governance process for the new network. This resulted in the design outlined by TIP-2, which was voted and approved by the two original communities.

Lifecycle of a successful proposal

1. **Forum discussion**: A Threshold community member posts a potential proposal on the Threshold governance forums. Community members who post the initial proposal are encouraged to get active in Meta governance discussions within the community on forums and in Discord to earn support for the proposal. There is no minimum token threshold required to create a new proposal on the forums. Community mods reserve the right to moderate proposals during this stage by simply deleting the proposal from the forums.
2. **Temperature Check:** Once a potential proposal has enough traction and discussion within the community, it can proceed for a temperature check, which requires off-chain snapshots to pass. If a proposal receives majority support during the temperature check snapshot, it is eligible to move onto the next step.
3. **Proposal Creation**: A community member holding enough vote weight submits the proposal on-chain. The proposal enters a **2-day delay period before voting** officially begins.
4. **Vote Period**: Proposal **voting remains open for 10 days**. If the proposal passes with enough quorum, it moves onto the next step; if the proposal fails, it is canceled. Creators and supporters of the proposal may bring a modified proposal forward again but it must be sufficiently different and pass requirements outlined in previous steps.
5. **Timelock Period**: Once a proposal is approved, the Governor smart contracts include an additional **timelock delay of 2 days**. Anyone can interact with the Governor smart contracts to queue an approved proposal into the Timelock contract.
6. **Execution**: After the timelock delay, anyone can execute an approved proposal.

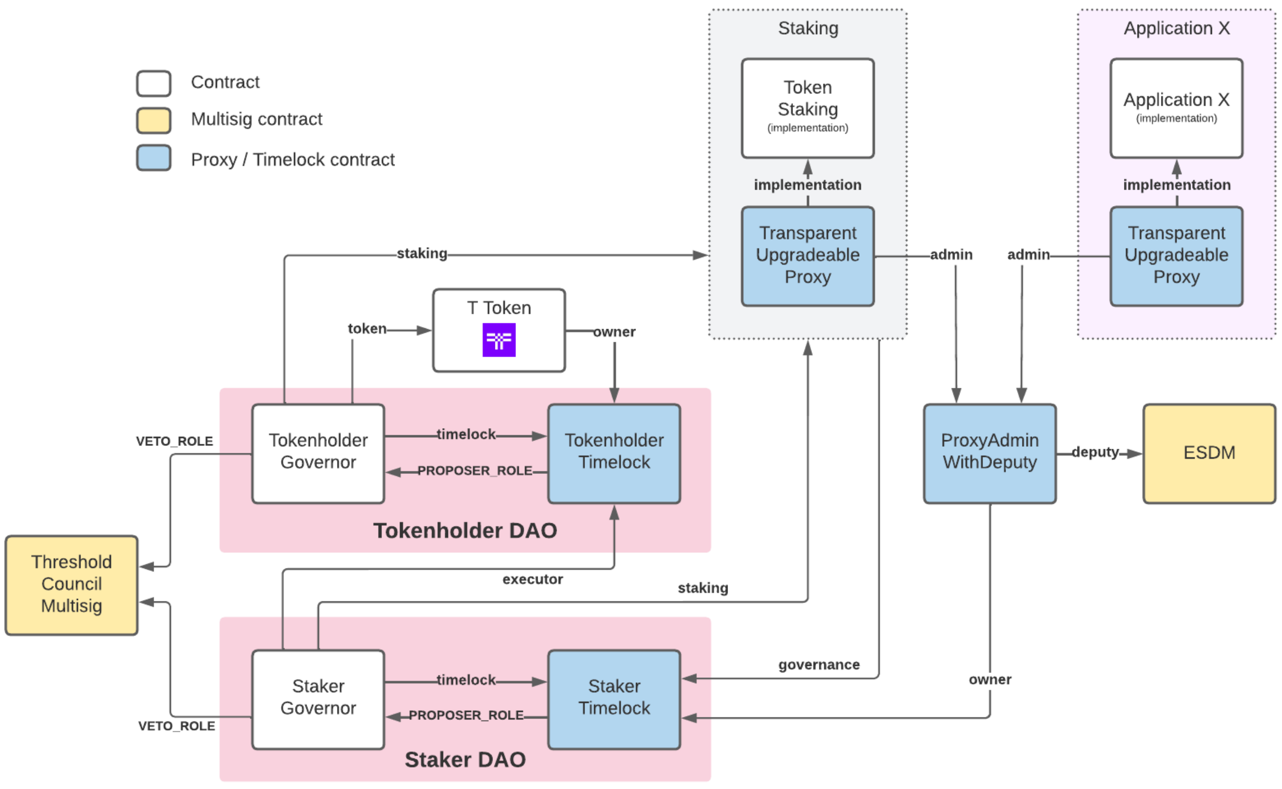


The lifecycle of a successful proposal

Deviations in the proposal lifecycle

- **Council vetos:** During the on-chain phase, the Elected Council can veto any proposal. This is intended to be an extra security mechanism in the event that a dangerous proposal passes.
- **Late quorum prevention:** The 10-days voting period is automatically extended when quorum is reached late, to prevent governance attacks that try to reach quorum at the last minute; in case a proposal reaches quorum less than 2 days before the deadline, the proposal deadline is **extended for 2 more days** from the moment the quorum was reached.

🏛️ Governor



Contract
Multisig contract
Proxy / Timelock contract
Staking
Token Staking (implementation)
implementation
Transparent Upgradeable Proxy
Application X
Application X (implementation)
implementation
Transparent Upgradeable Proxy
admin
admin
staking
T Token
token
owner
VETO_ROLE
Tokenholder Governor
timelock
PROPOSER_ROLE
Tokenholder Timelock
Tokenholder DAO
ProxyAdmin WithDeputy
deputy
ESDM
Threshold Council Multisig
executor
staking
governance
owner
Staker Governor
timelock
PROPOSER_ROLE
Staker Timelock
VETO_ROLE
Staker DAO

⚡ Snapshots

🙋 ♀ ♀ Guilds

Additionally to the DAO governance bodies, there are community-led guilds such as the Marketing Guild, the Integrations Guild, and the Treasury Guild. Each guild is managed by an elected committee and holds regular, rotating elections.

Join a guild (via Discord - `#dao-contribute` channel) and work together with other Threshold DAO members based on your interests and expertise!

Treasury Guild

The Threshold Treasury Guild is responsible for effectively managing the Threshold DAO treasury. This includes growing Protocol Owned Liquidity (POL), researching / implementing best practice treasury management strategies, executing ecosystem liquidity incentives, diversifying the treasury, etc.

Integrations Guild

The core mission of the Threshold Integrations Guild is to build successful, synergistic and long-lasting relationships with other protocols, DAOs and external organizations.

Marketing Guild

The Threshold Marketing Guild is responsible for general Threshold marketing across services, growing our network of contributors, onboarding new members to the Threshold DAO, educating people about the Threshold's value, services and use cases, and more.

💰 Threshold Multisigs

Compilation of current multisigs for the Threshold council and guilds

Council Multisig

Gnosis Safe

Council Multisig at Gnosis Safe

ⓘ The Threshold Council multisig is set to **6 of 9**.

Guilds Multisigs

Treasury Guild Multisig

Gnosis Safe

Treasury Guild Multisig at Gnosis Safe

ⓘ This multisig is set to **6 of 9** and can be changed by the guild committee if needed.

Integrations Guild Multisig

Gnosis Safe

Integrations Guild Multisig at Gnosis Safe

ⓘ This multisig is set to **3 of 5** and can be changed by the guild committee if needed.

Marketing Guild Multisig

Gnosis Safe

Marketing Guild Multisig at Gnosis Safe

This multisig is set to **5 of 8** and can be changed by the guild committee if needed.

ESDM - Emergency Security Developer Multisig

To be created

The Threshold ESDM multisig will be set to **3 of 4**.

DAO Vote Delegation

To participate in DAO governance, stakers or token holders must set a governance delegate address. Vote delegation is the process of granting a delegate the power to vote on your behalf, using your voting weight, on DAO governance issues. The delegate can be yourself (self-delegation) or a third party. A configured delegate can be changed or revoked at any time.

> ⓘ Delegation **never involves the transfer of custody of assets** but rather just the vote weight those assets represent in the Threshold DAO.

Third-Party Delegation

Third-party delegates are volunteers who actively participate in Threshold governance and wish to grow their influence over critical DAO decisions. Delegates provide ETH addresses to which other Threshold token holders and Threshold stakers can delegate their vote weight. Third-party delegates are subsequently responsible for voting on Threshold governance proposals with this vote weight that others have assigned to them.

It is ideal for those who are unsure or uninterested in governance and does not want to participate in proposals, discussions, and voting. Instead, they can select active DAO members who have demonstrated a commitment to Threshold to make decisions on their behalf using their token weight and stake weight. Delegating their vote allows token holders and stakers to have an indirect say in the DAO without being involved in day-to-day governance activities.

> ⓘ There is no established compensation for third-party delegate contribution to governance.

Token Weight Delegation

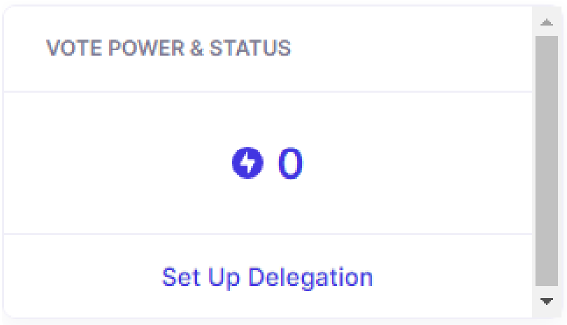
Liquid T holders who aren't staking on Threshold can delegate their token weights to themselves or a third party for voting on governance proposals.

> ⓘ Delegation is accomplished via an on-chain transaction which costs ETH.

1. Go to https://boardroom.io/threshold

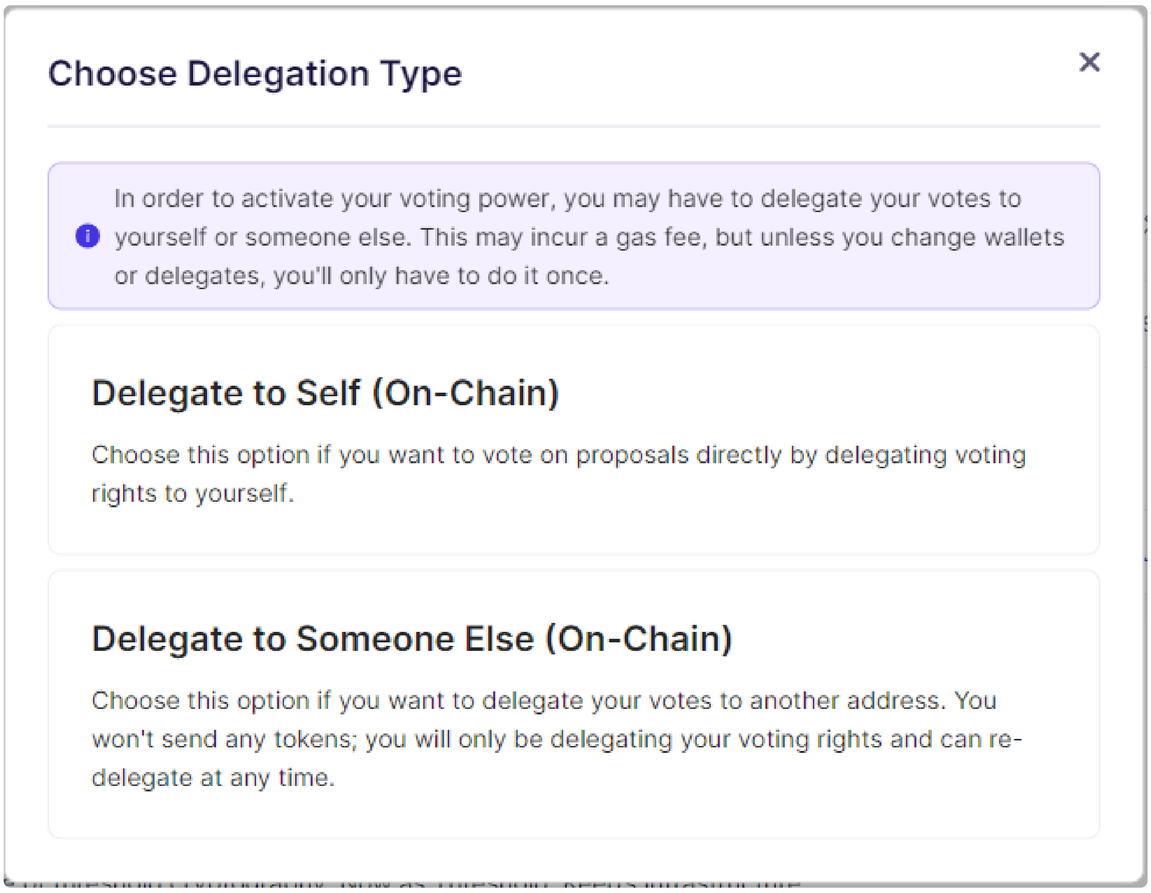
2. Connect your wallet

3. Click "*Set Up Delegation"*



Set Up Delegation

4. Select your *"Delegation Type"* - either to yourself or a third party.


Choose Delegation Type

In order to activate your voting power, you may have to delegate your votes to yourself or someone else. This may incur a gas fee, but unless you change wallets or delegates, you'll only have to do it once.

Delegate to Self (On-Chain)

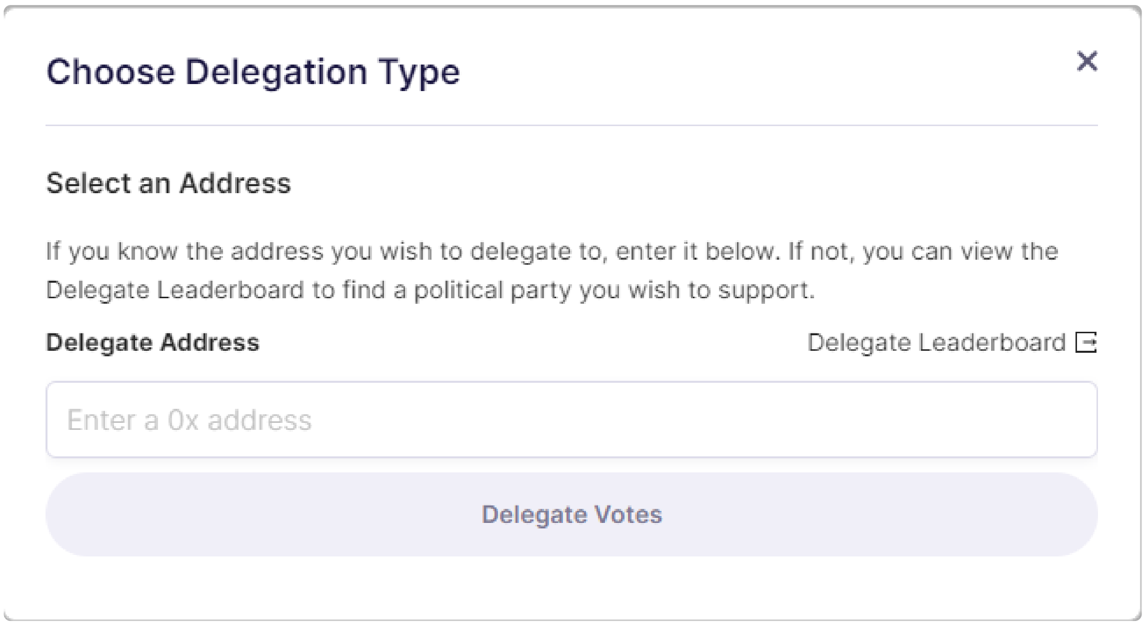
Choose this option if you want to vote on proposals directly by delegating voting rights to yourself.

Delegate to Someone Else (On-Chain)

Choose this option if you want to delegate your votes to another address. You won't send any tokens; you will only be delegating your voting rights and can re-delegate at any time.

Choose Delegation Type

5. Enter *"Delegate Address"* and click on *"Delegate Votes"*


Choose Delegation Type

Select an Address

If you know the address you wish to delegate to, enter it below. If not, you can view the Delegate Leaderboard to find a political party you wish to support.

Delegate Address

Delegate Votes

6. Sign the transaction

Stake Weight Delegation

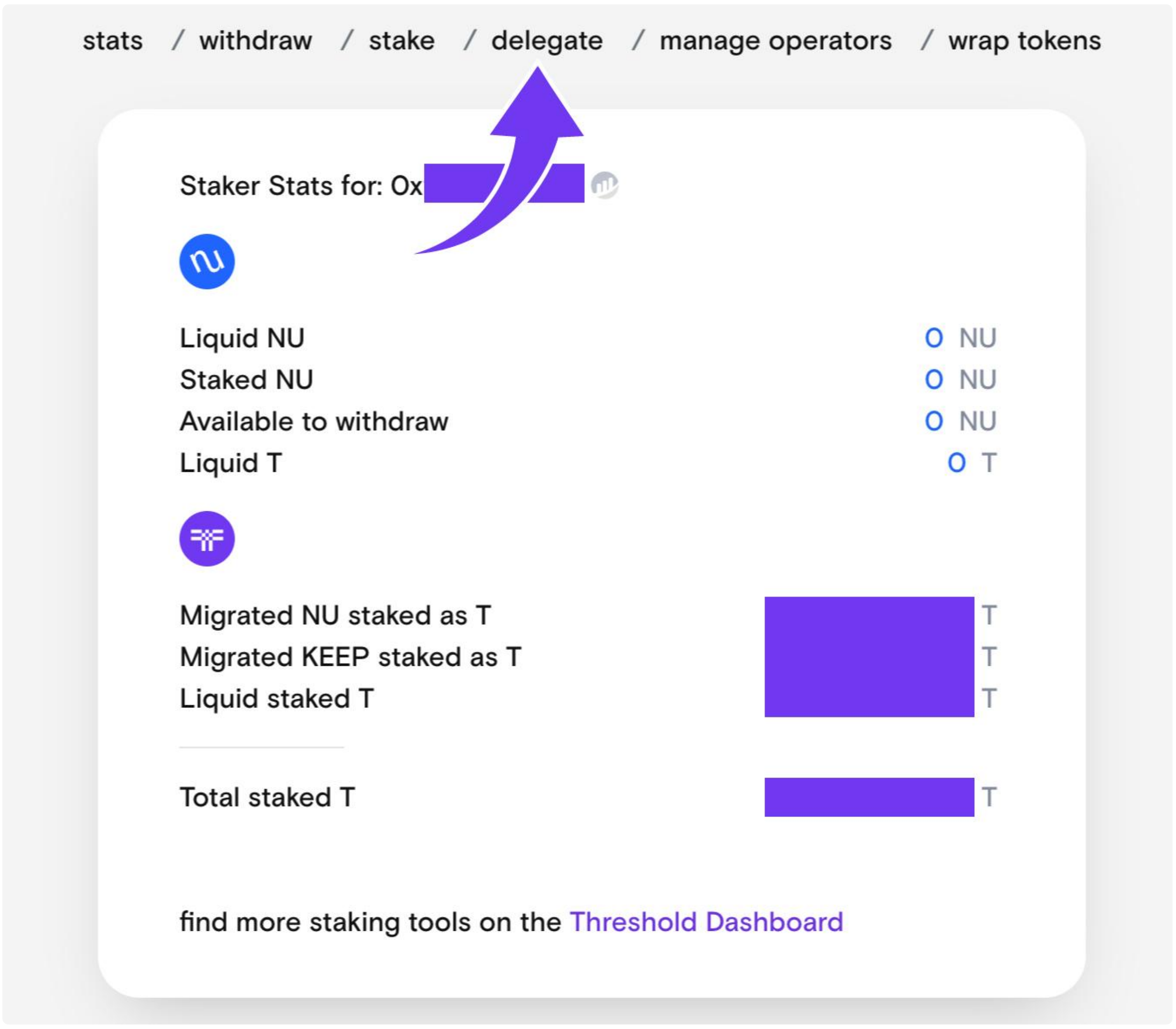
Stakers on Threshold can delegate their stake weights to themselves or a third party for voting on governance proposals.

> ⓘ Delegation is accomplished via an on-chain transaction which costs ETH.

1. Go to https://stake.nucypher.network/manage

2. Connect your wallet

3. Click on the *"delegate"* tab in the navigation bar


stats / withdraw / stake / delegate / manage operators / wrap tokens

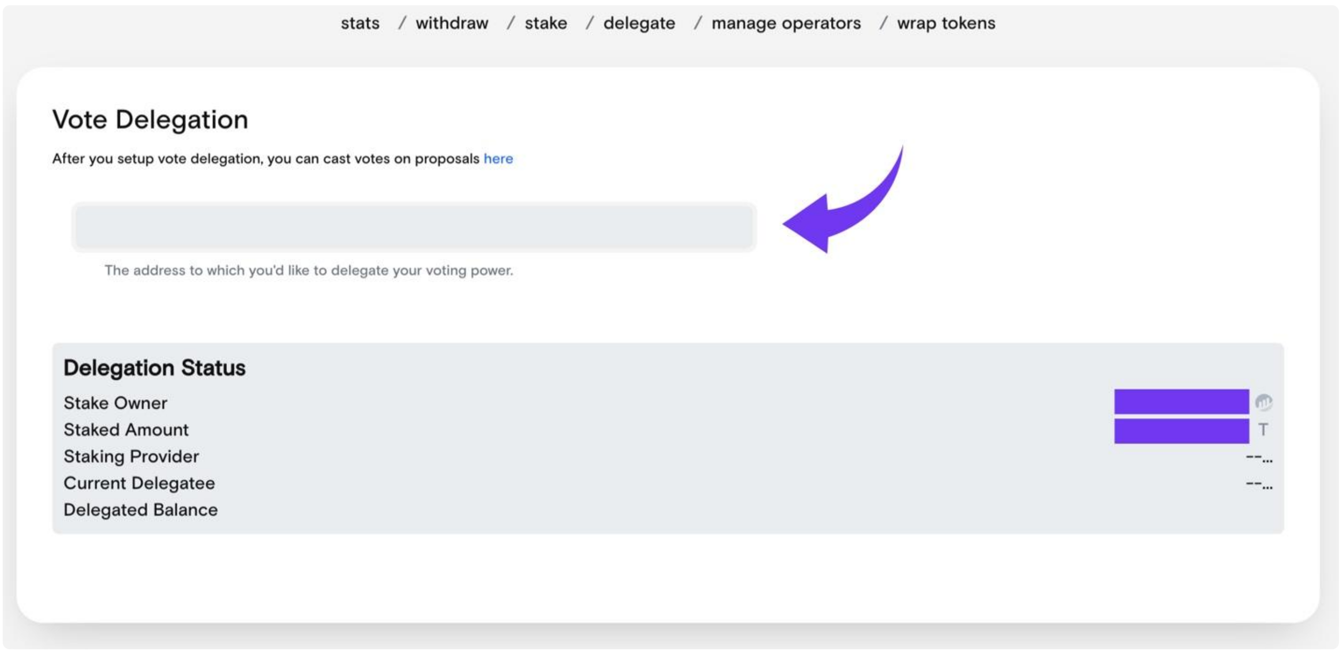
Staker Stats for: 0x[redacted]

Liquid NU	0 NU
Staked NU	0 NU
Available to withdraw	0 NU
Liquid T	0 T
Migrated NU staked as T	[redacted] T
Migrated KEEP staked as T	[redacted] T
Liquid staked T	[redacted] T
Total staked T	[redacted] T

find more staking tools on the Threshold Dashboard

"delegate" Tab

4. Enter the delegate address - once an address is entered, a *"Delegate"* button will appear


stats / withdraw / stake / delegate / manage operators / wrap tokens

Vote Delegation

After you setup vote delegation, you can cast votes on proposals here

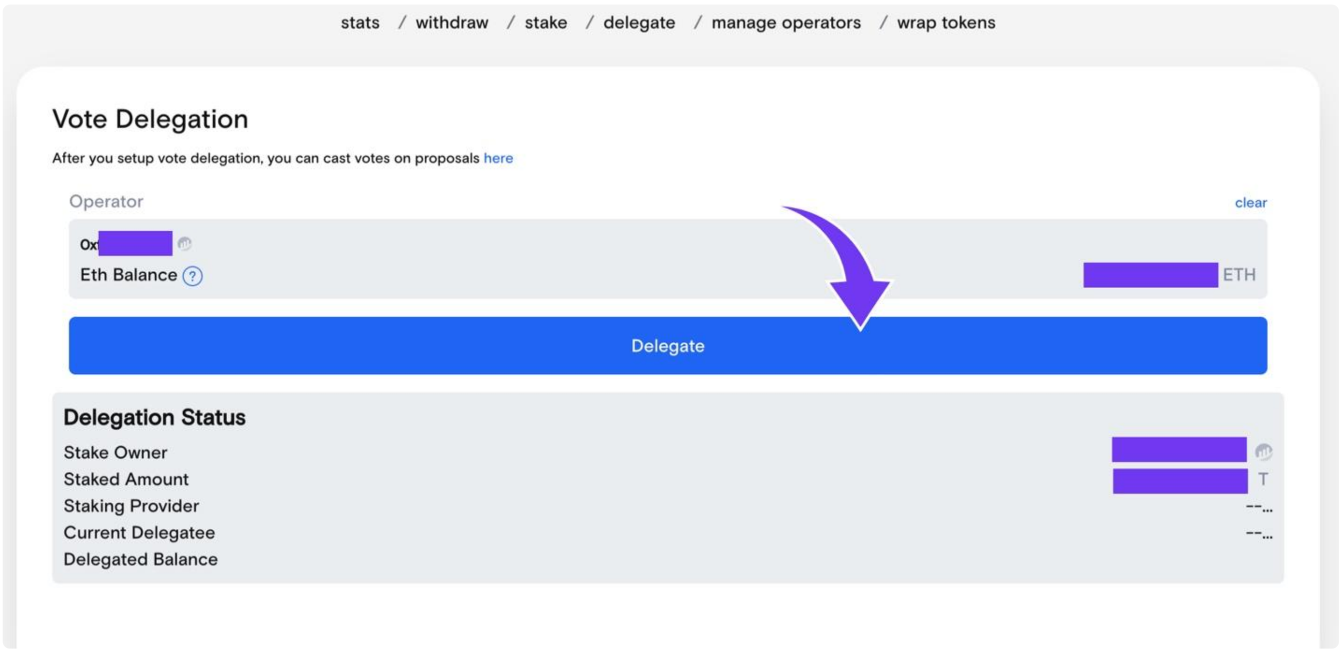
The address to which you'd like to delegate your voting power.

Delegation Status

Stake Owner	[redacted]
Staked Amount	[redacted] T
Staking Provider	--...
Current Delegatee	--...
Delegated Balance	

Enter Delegate Address

5. Click *"Delegate"* and sign the transaction.


stats / withdraw / stake / delegate / manage operators / wrap tokens

Vote Delegation

After you setup vote delegation, you can cast votes on proposals here

Operator clear

0x[redacted]

Eth Balance ? [redacted] ETH

Delegate

Delegation Status

Stake Owner	[redacted]
Staked Amount	[redacted] T
Staking Provider	--...
Current Delegatee	--...
Delegated Balance	

Delegate Button

Staking & Running a Node

Upgrade NU & KEEP to T

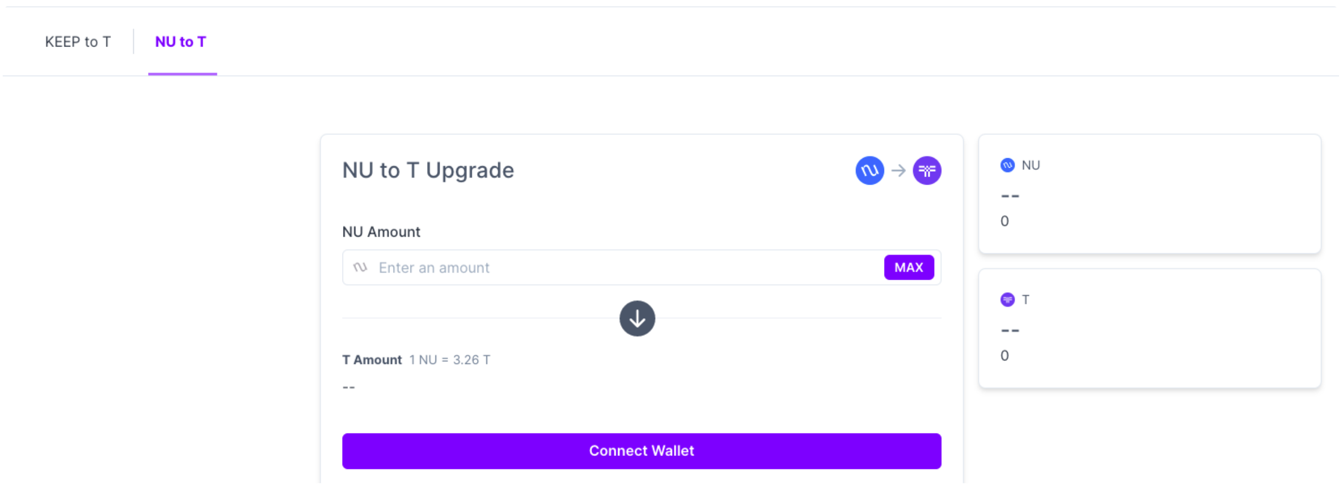
NU and KEEP tokens can be upgraded to T tokens via the Vending Machine contracts.

As decided by the NU and KEEP communities in the final merge proposal, the conversion ratio for each token is based on the total supply rather than price. The final total supply of NU is ~1,380,688,920 and the total supply of KEEP is ~940,795,010.

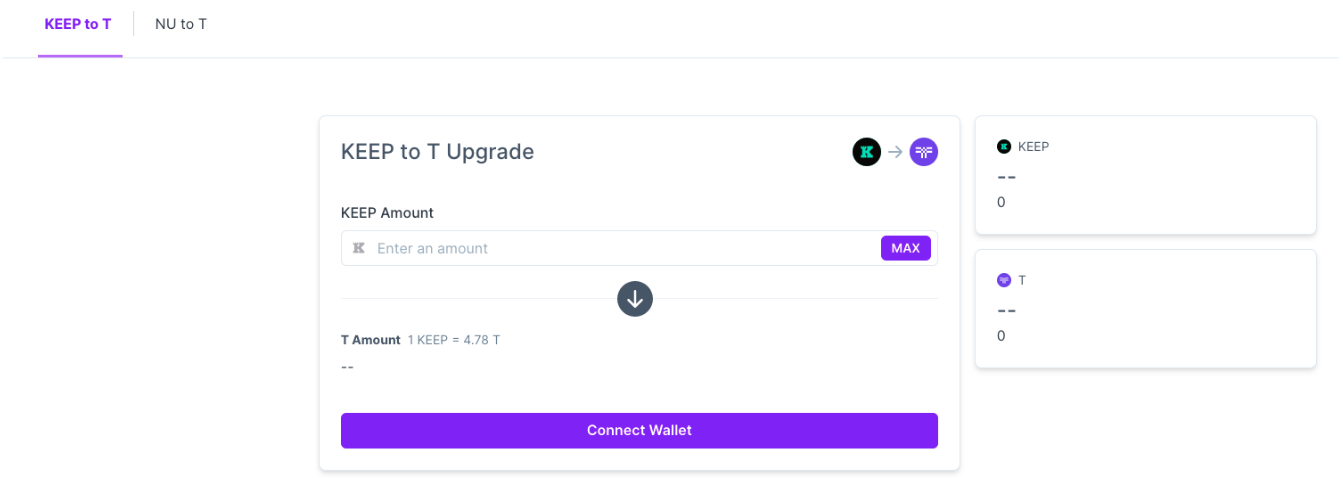
As a result, the token factors are:

- 1 NU: ~3.26 T
- 1 KEEP: ~4.78 T

Token upgrades can be performed via the Threshold Dashboard which interfaces directly with the Vending Machine smart contracts.



NU->T Upgrade



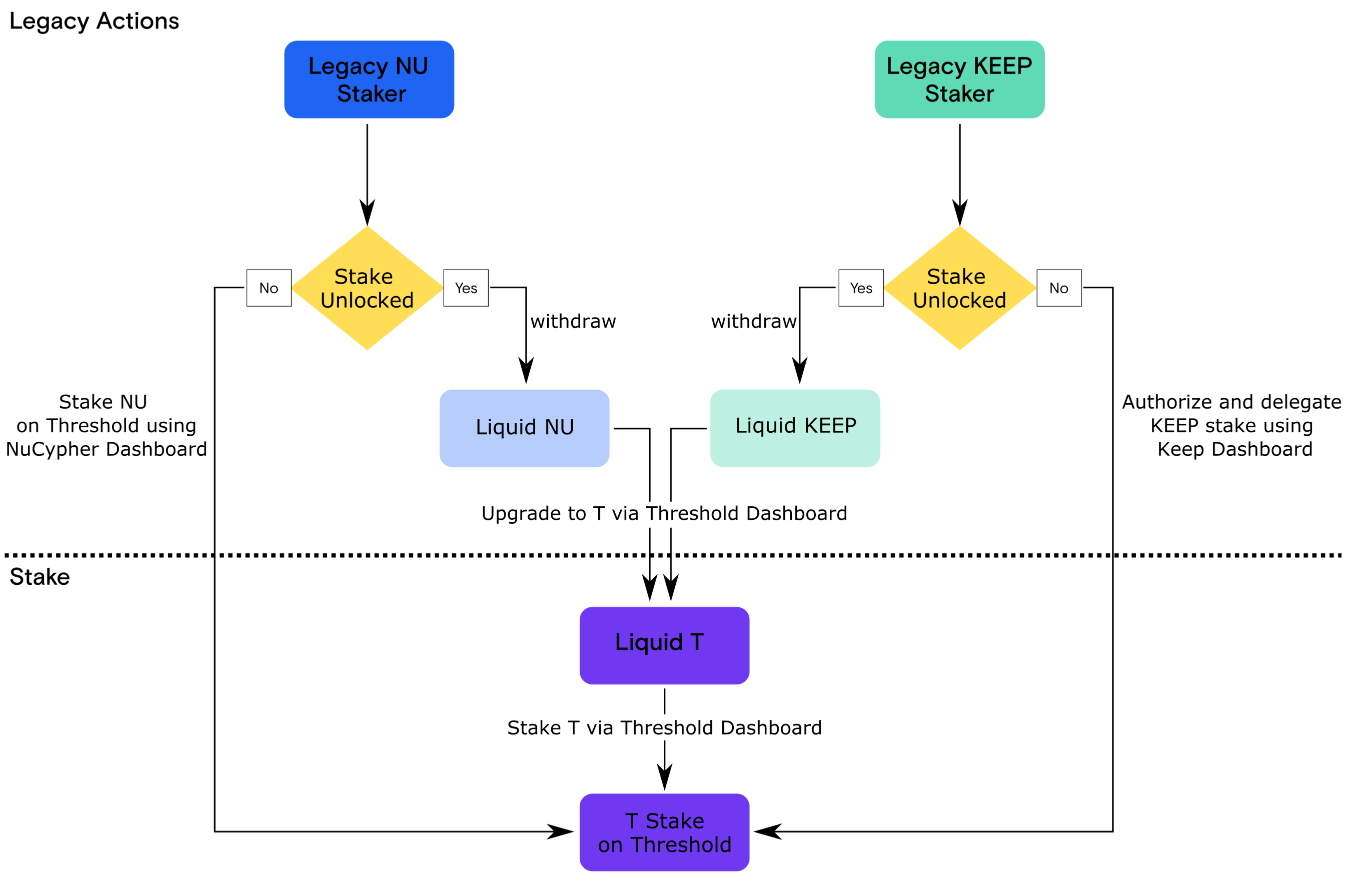
Keep->T Upgrade

Staking on Threshold

Stakes on Threshold can take the following forms:

- T stake using liquid T created on the Threshold Dashboard
- Legacy NU stake migrated to T from the NuCypher Dashboard
- Legacy KEEP stake migrated to T from the Keep Dashboard

Here is a summary of the overall process:


Legacy Actions
Legacy NU Staker
Legacy KEEP Staker
Stake Unlocked
No
Yes
withdraw
Stake Unlocked
Yes
No
withdraw
Stake NU on Threshold using NuCypher Dashboard
Liquid NU
Liquid KEEP
Authorize and delegate KEEP stake using Keep Dashboard
Upgrade to T via Threshold Dashboard
Stake
Liquid T
Stake T via Threshold Dashboard
T Stake on Threshold

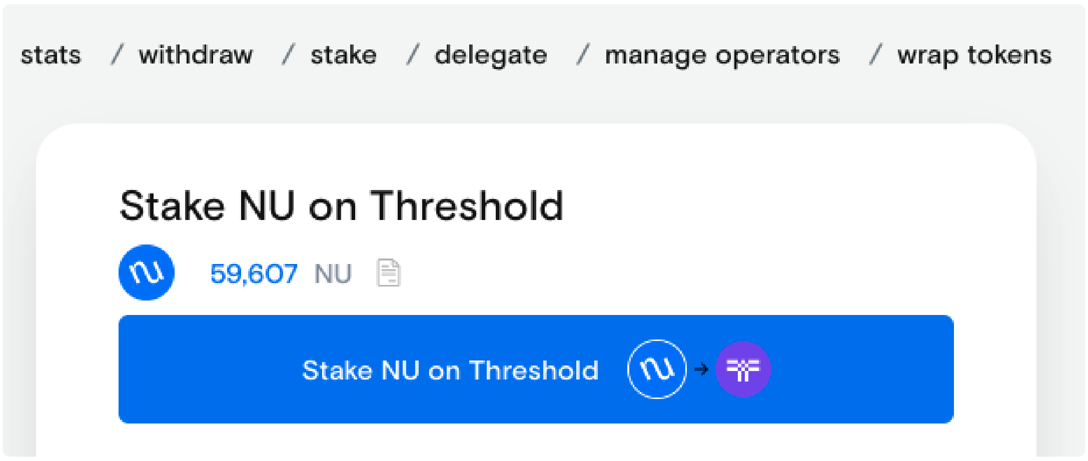
Stake Creation

Stake Liquid T

- Go to https://dashboard.threshold.network/staking
- Connect your wallet
- Your T balance will be shown - click on *"Stake"*
- There will be a pop-up describing the staking process; click the check-box to acknowledge you have read the requirements
- Choose an amount of T to stake and confirm the transaction

Migrate Legacy NU Stake

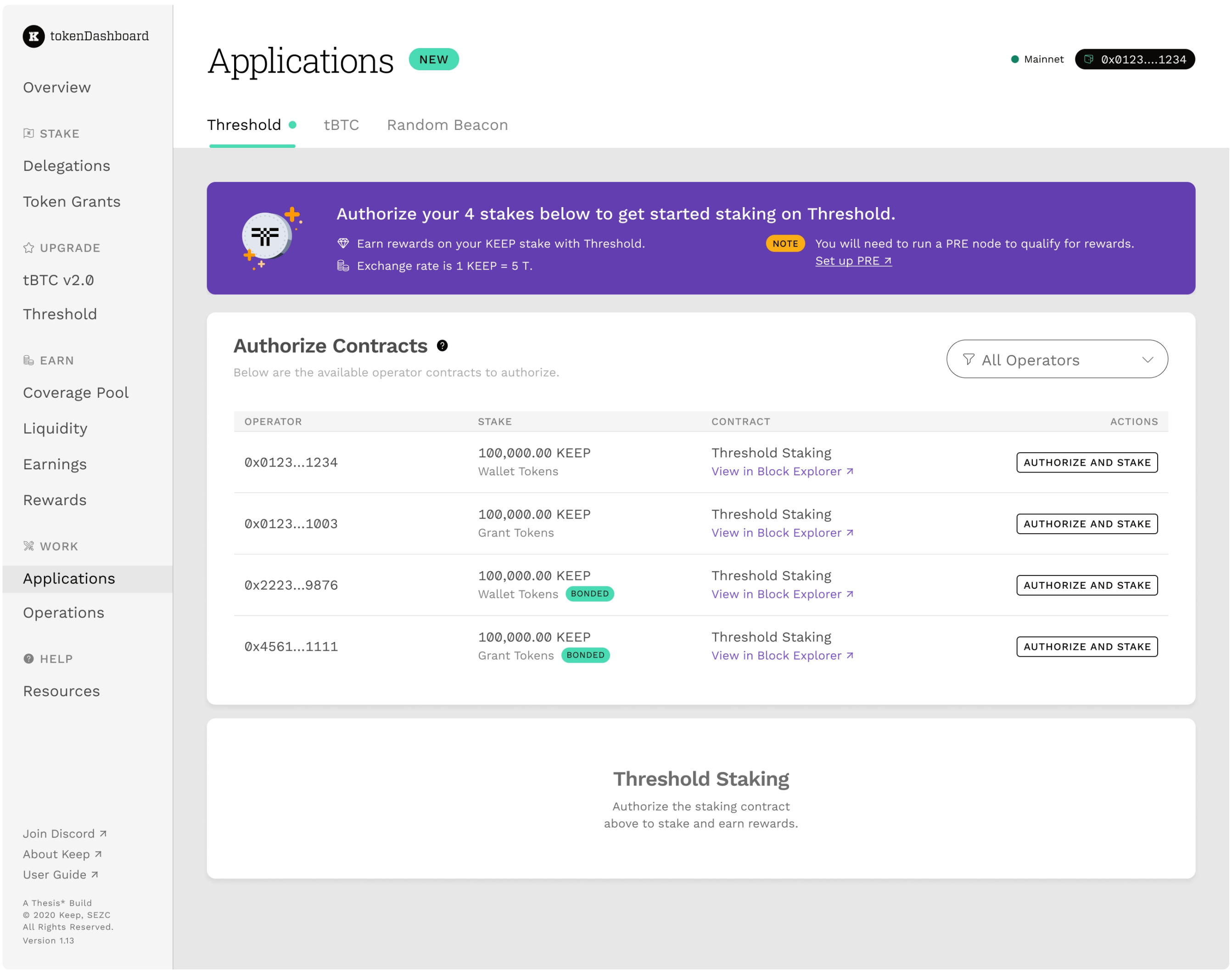
- Go to https://stake.nucypher.network/manage/stake
- Click *"Stake NU on Threshold"*


stats / withdraw / stake / delegate / manage operators / wrap tokens
Stake NU on Threshold
59,607 NU
Stake NU on Threshold

Stake NU on Threshold

Migrate Legacy KEEP Stake

- Authorize Threshold Staking for the Keep Network legacy stake - https://dashboard.keep.network/applications/threshold



tokenDashboard

Overview

STAKE

Delegations

Token Grants

UPGRADE

tBTC v2.0

Threshold

EARN

Coverage Pool

Liquidity

Earnings

Rewards

WORK

Applications

Operations

HELP

Resources

Join Discord ↗
About Keep ↗
User Guide ↗

A Thesis* Build
© 2020 Keep, SEZC
All Rights Reserved.
Version 1.13

Applications NEW

● Mainnet 0x0123....1234

Threshold ● tBTC Random Beacon

Authorize your 4 stakes below to get started staking on Threshold.

Earn rewards on your KEEP stake with Threshold.
Exchange rate is 1 KEEP = 5 T.

NOTE You will need to run a PRE node to qualify for rewards. Set up PRE ↗

Authorize Contracts

Below are the available operator contracts to authorize.

All Operators

OPERATOR	STAKE	CONTRACT	ACTIONS
0x0123...1234	100,000.00 KEEP Wallet Tokens	Threshold Staking View in Block Explorer ↗	AUTHORIZE AND STAKE
0x0123...1003	100,000.00 KEEP Grant Tokens	Threshold Staking View in Block Explorer ↗	AUTHORIZE AND STAKE
0x2223...9876	100,000.00 KEEP Wallet Tokens BONDED	Threshold Staking View in Block Explorer ↗	AUTHORIZE AND STAKE
0x4561...1111	100,000.00 KEEP Grant Tokens BONDED	Threshold Staking View in Block Explorer ↗	AUTHORIZE AND STAKE

Threshold Staking

Authorize the staking contract above to stake and earn rewards.

Stake KEEP on Threshold

Running a Node

Get involved in staking the Threshold Network

Have what it takes to run a node of your own?

Click here for a guide on how to setup your own PRE node.

Click here to learn how to setup a tBTC v2 node.

> Please be aware that running a node is not an easy task and requires technical skill and commitment to maintaining node uptime and availability.

Prefer staking with Professional Staking Providers instead?

Click here to see a list of professional grade staking providers.

> ***Staking Providers have not been vetted or endorsed by Threshold. Use your judgement when selecting a provider.***

Self-Managed

PRE node setup with `nucypher-ops`

`nucypher-ops` is an ansible script that automates setup and configuration of a PRE node.

nucypher-ops setup guide

tBTC v2 Client Setup

A guide for to setup and configure the tBTC v2 client.

tBTC v2 Client Setup

PRE Node Setup

This document provides basic instructions for use of the nucypher-ops utility for setting up PRE nodes.

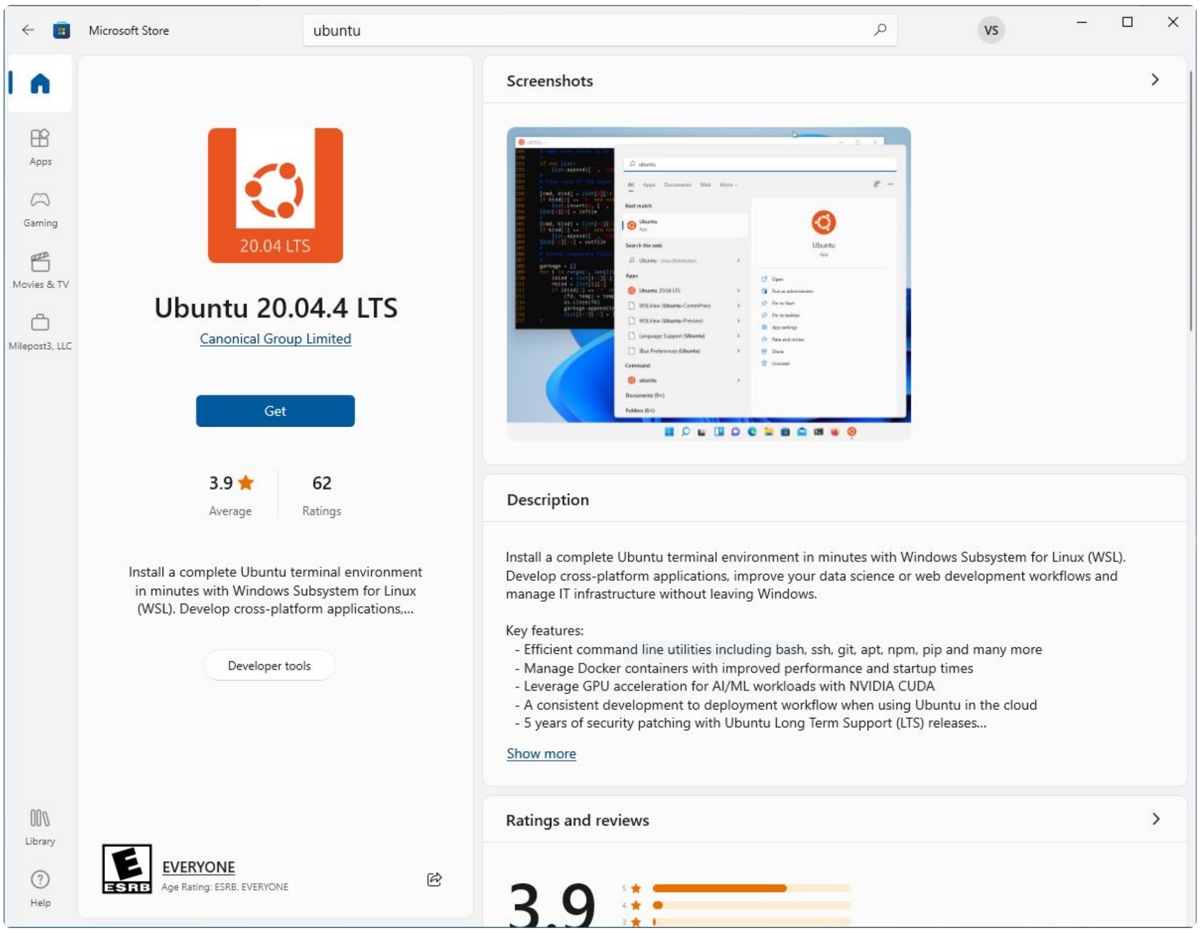
This document will help you get a node running to which you can bond your stake. **Please note that running a node is not a passive activity**.

Nucypher-ops is an ansible script that will setup and configure a PRE node for you. Currently, the script supports AWS and Digital Ocean. This document will guide you through the setup process using visuals from DigitalOcean.

Local Machine Considerations

Windows

Install Ubuntu 20.04.4 LTS through the Windows Store:



Install a complete Ubuntu terminal environment in minutes with Windows Subsystem for Linux (WSL). Develop cross-platform applications,...

Install a complete Ubuntu terminal environment in minutes with Windows Subsystem for Linux (WSL). Develop cross-platform applications, improve your data science or web development workflows and manage IT infrastructure without leaving Windows.

Key features:
- Efficient command line utilities including bash, ssh, git, apt, npm, pip and many more
- Manage Docker containers with improved performance and startup times
- Leverage GPU acceleration for AI/ML workloads with NVIDIA CUDA
- A consistent development to deployment workflow when using Ubuntu in the cloud
- 5 years of security patching with Ubuntu Long Term Support (LTS) releases...

WSL is the most convenient way to run nucypher-ops on a Windows based local machine

To begin, update your local machine and install dependencies

```
sudo apt update
sudo apt upgrade
sudo apt-get install libffi-dev python3-dev python3-pip python3-virtualenv build-
essential libssl-dev
```

Read prompts and respond appropriately. This will take a few minutes.

MacOS

Install Python and pip on your local machine, if you don't have it already:

Python Website

Visit the Python.org Website

PyPA Website

Visit the pypa.io Website and follow the steps provided to make sure that you have a working Python with pip installed.

Linux

To begin, update your local machine and install dependencies

```
sudo apt update
sudo apt upgrade
sudo apt-get install libffi-dev python3-dev python3-pip python3-virtualenv build-essential libssl-dev
```

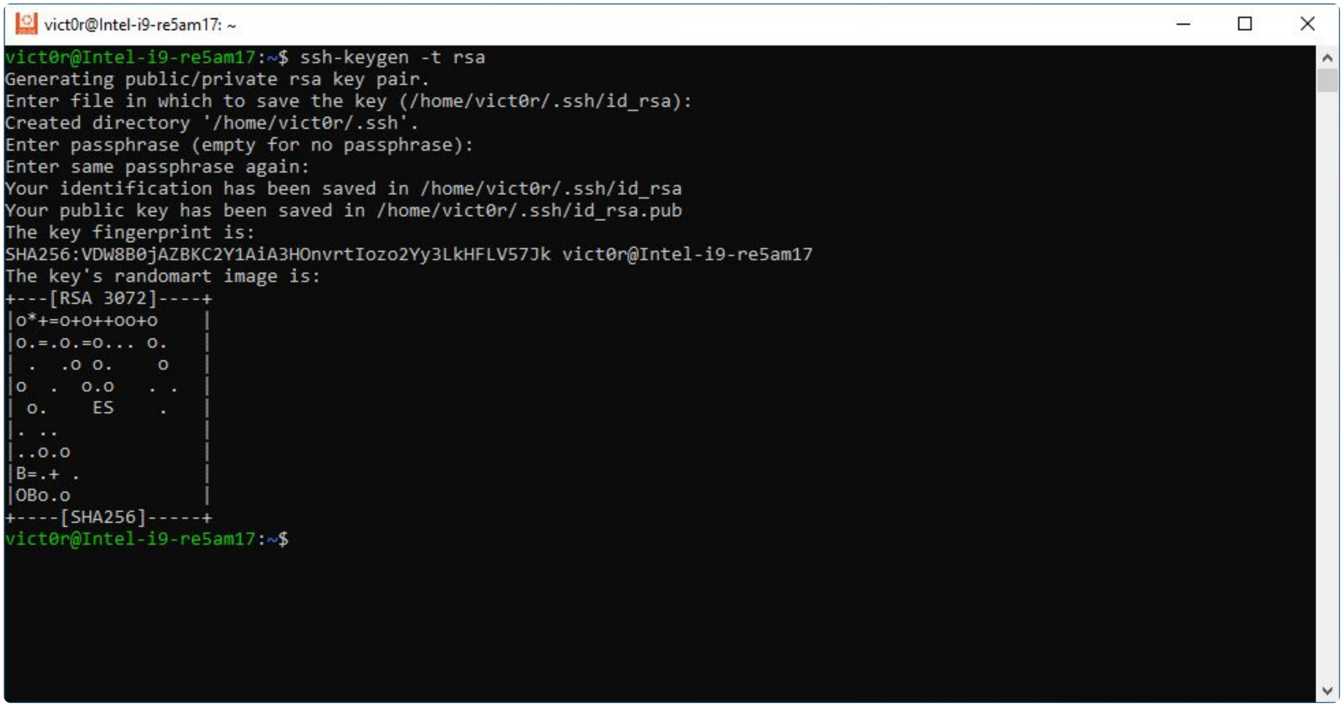
Read prompts and respond appropriately. This will take a few minutes.

SSH Key

Generate an SSH key pair. We will need the **public key**.

Visit the OpenSSH website to learn about OpenSSH.

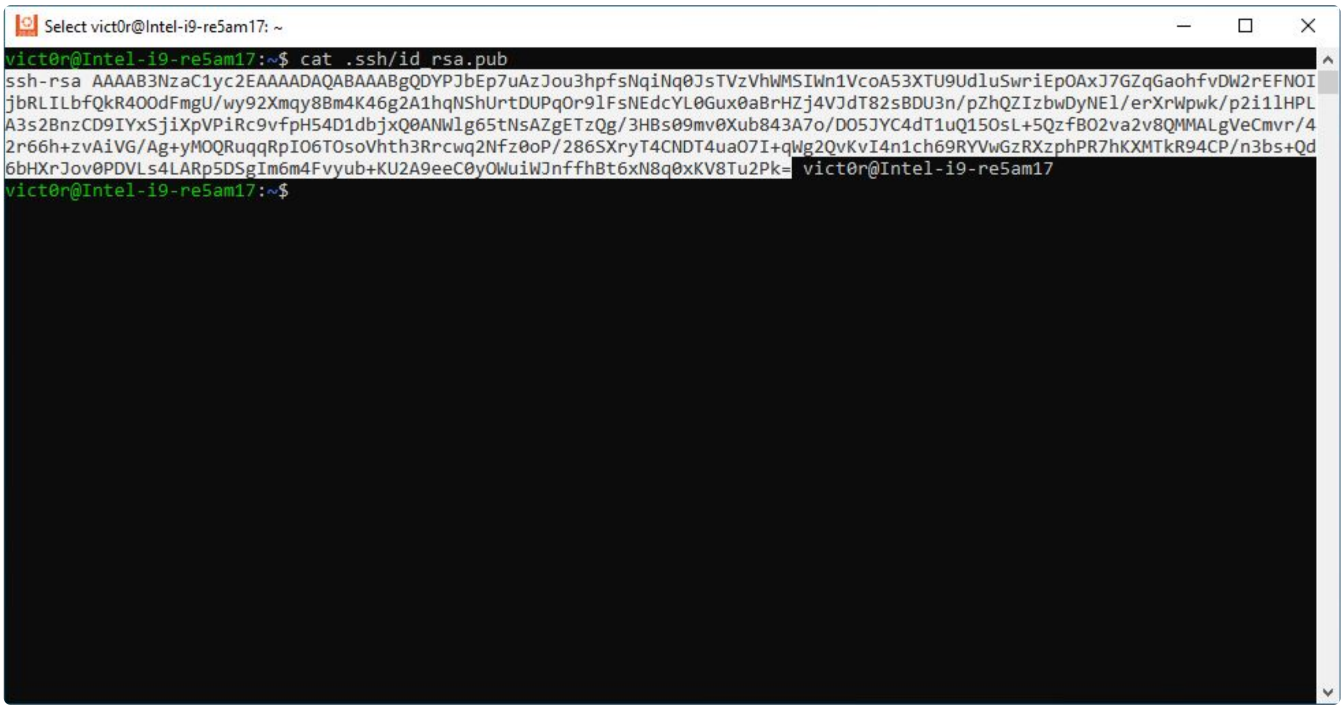
```
ssh-keygen -t rsa
 cat .ssh/id_rsa.pub
```



```
vict0r@Intel-i9-re5am17:~$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/vict0r/.ssh/id_rsa):
Created directory '/home/vict0r/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/vict0r/.ssh/id_rsa
Your public key has been saved in /home/vict0r/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:VDW8B0jAZBKC2Y1AiA3HOnvrtIozo2Yy3LkHFLV57Jk vict0r@Intel-i9-re5am17
The key's randomart image is:
+---[RSA 3072]----+
|o*+=o+o++oo+o    |
|o.=.o.=o... o.   |
| .  .o o.    o   |
|o  .  o.o   . .  |
| o.    ES    .   |
|. ..             |
|..o.o            |
|B=.+ .           |
|OBo.o            |
+----[SHA256]-----+
vict0r@Intel-i9-re5am17:~$
```

Generating a new SSH Key

Copy the highlighted part

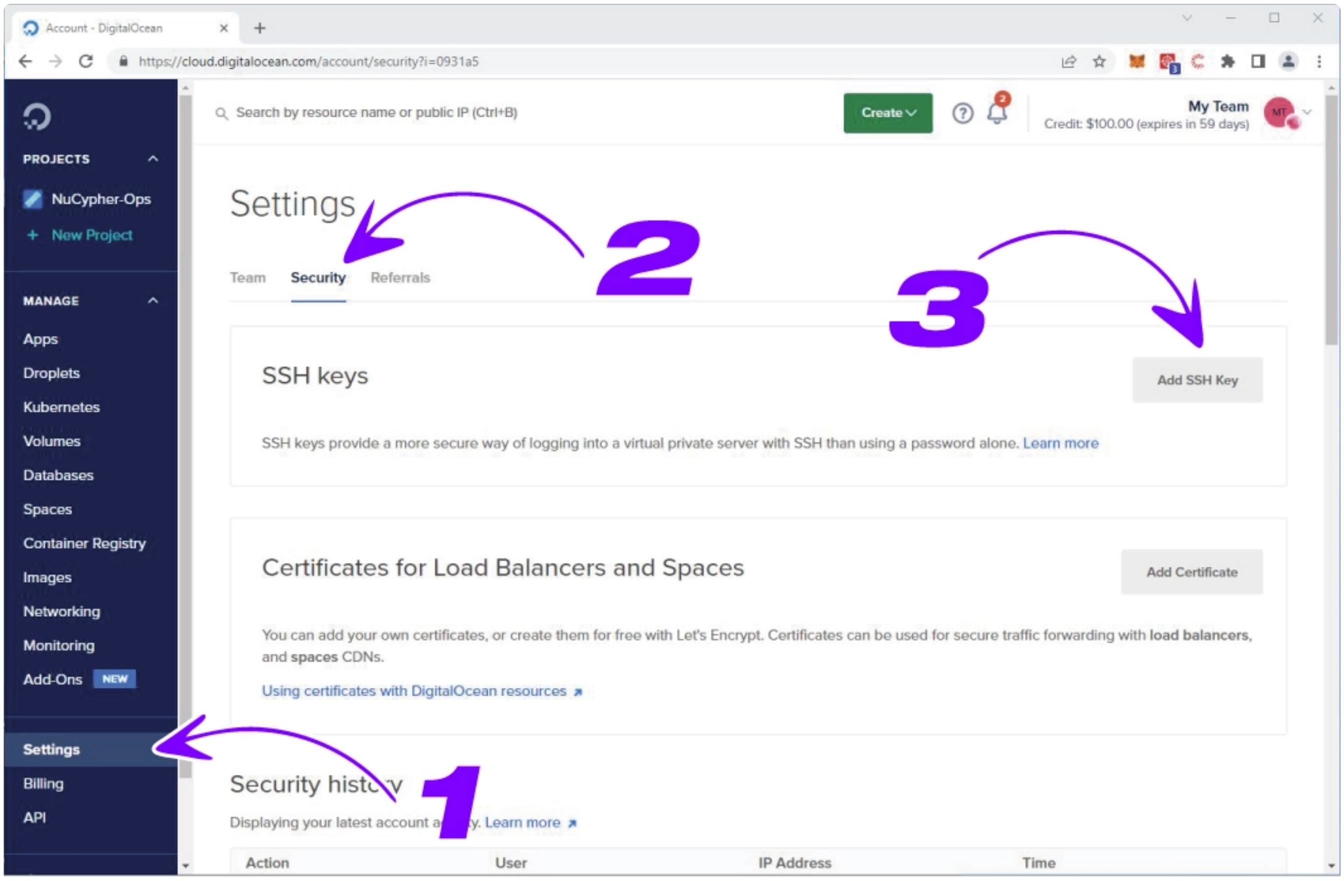


```
vict0r@Intel-i9-re5am17:~$ cat .ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQDYPJbEp7uAzJou3hpfsNqiNq0JsTVzVhWMSIWn1VcoA53XTU9UdluSwriEpOAxJ7GZqGaohfvDW2rEFNOI
jbRLILbfQkR4OOdFmgU/wy92Xmqy8Bm4K46g2A1hqNShUrtDUPqOr9lFsNEdcYL0Gux0aBrHZj4VJdT82sBDU3n/pZhQZIzbwDyNEl/erXrWpwk/p2i1lHPL
A3s2BnzCD9IYxSjiXpVPiRc9vfpH54D1dbjxQ0ANWlg65tNsAZgETzQg/3HBs09mv0Xub843A7o/DO5JYC4dT1uQ15OsL+5QzfBO2va2v8QMMALgVeCmvr/4
2r66h+zvAiVG/Ag+yMOQRuqqRpIO6TOsoVhth3Rrcwq2Nfz0oP/286SXryT4CNDT4uaO7I+qWg2QvKvI4n1ch69RYVwGzRXzphPR7hKXMTkR94CP/n3bs+Qd
6bHXrJov0PDVLs4LARp5DSgIm6m4Fvyub+KU2A9eeC0yOWuiWJnffhBt6xN8q0xKV8Tu2Pk= vict0r@Intel-i9-re5am17
vict0r@Intel-i9-re5am17:~$
```

New SSH Key

In your Digital Ocean dashboard:

Go to **Settings** > **Security** and click the **Add SSH Key** button.



SSH keys

SSH keys provide a more secure way of logging into a virtual private server with SSH than using a password alone. Learn more

Certificates for Load Balancers and Spaces

You can add your own certificates, or create them for free with Let's Encrypt. Certificates can be used for secure traffic forwarding with **load balancers**, and **spaces** CDNs.

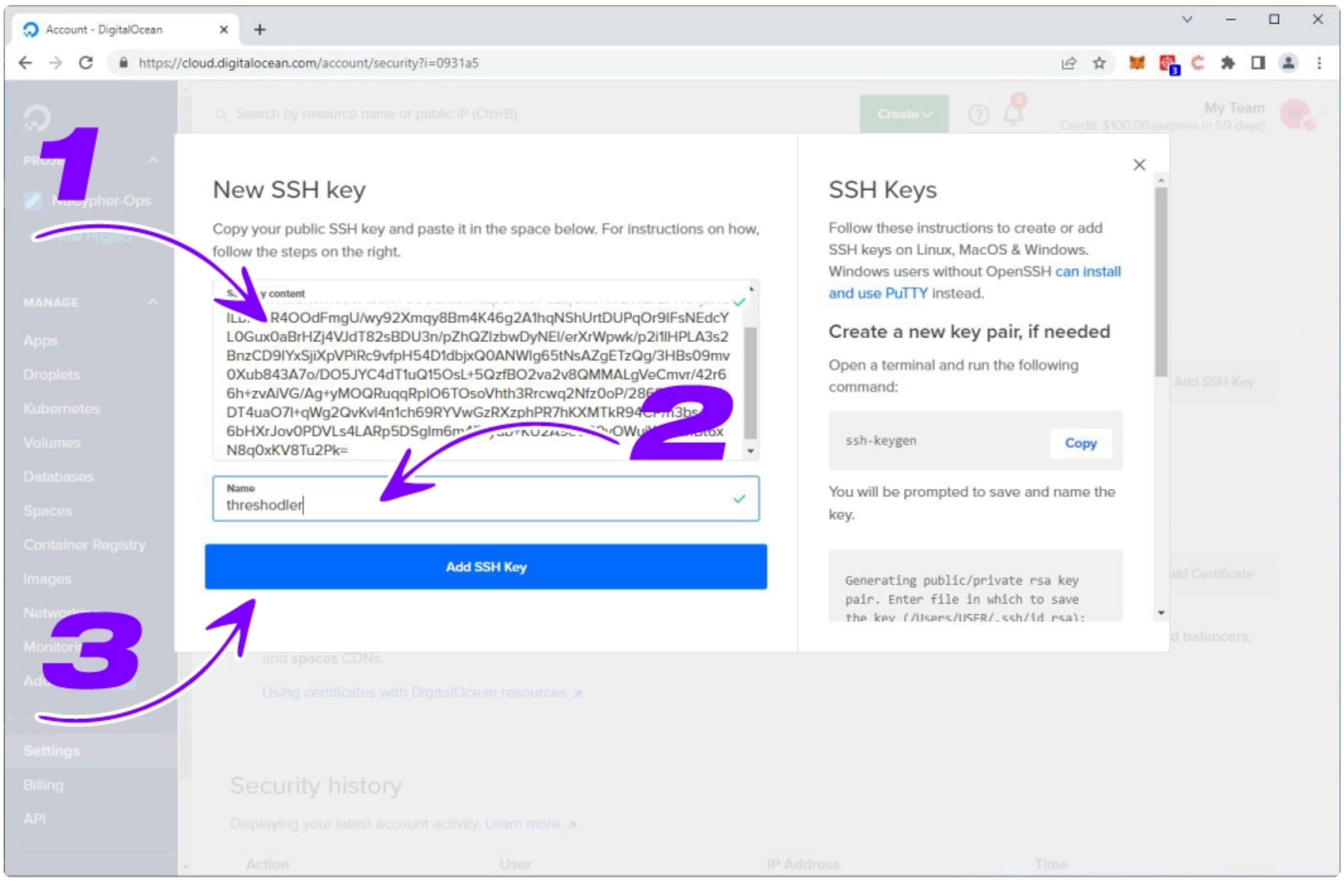
Using certificates with DigitalOcean resources

Security history

Displaying your latest account a[illegible]y. Learn more

Adding a new SSH Key

Paste the key into the box, give your key a name, and click the **Add SSH Key** button.



New SSH key

Copy your public SSH key and paste it in the space below. For instructions on how, follow the steps on the right.

SSH key content

```
ILL[illegible] R4OOdFmgU/wy92Xmqy8Bm4K46g2A1hqNShUrtDUPqOr9IFsNEdcY
L0Gux0aBrHZj4VJdT82sBDU3n/pZhQZIzbwDyNEl/erXrWpwk/p2i1lHPLA3s2
BnzCD9IYxSjiXpVPiRc9vfpH54D1dbjxQ0ANWIg65tNsAZgETzQg/3HBs09mv
0Xub843A7o/DO5JYC4dT1uQ15OsL+5QzfBO2va2v8QMMALgVeCmvr/42r6
6h+zvAiVG/Ag+yMOQRuqqRpIO6TOsoVhth3Rrcwq2Nfz0oP/28[illegible]
DT4uaO7I+qWg2QvKvl4n1ch69RYVwGzRXzphPR7hKXMTkR94[illegible]3bs
6bHXrJov0PDVLs4LARp5DSglm6m[illegible]OWu[illegible]
N8q0xKV8Tu2Pk=
```

Name: threshodler

SSH Keys

Follow these instructions to create or add SSH keys on Linux, MacOS & Windows. Windows users without OpenSSH can install and use PuTTY instead.

Create a new key pair, if needed

Open a terminal and run the following command:

```
ssh-keygen
```

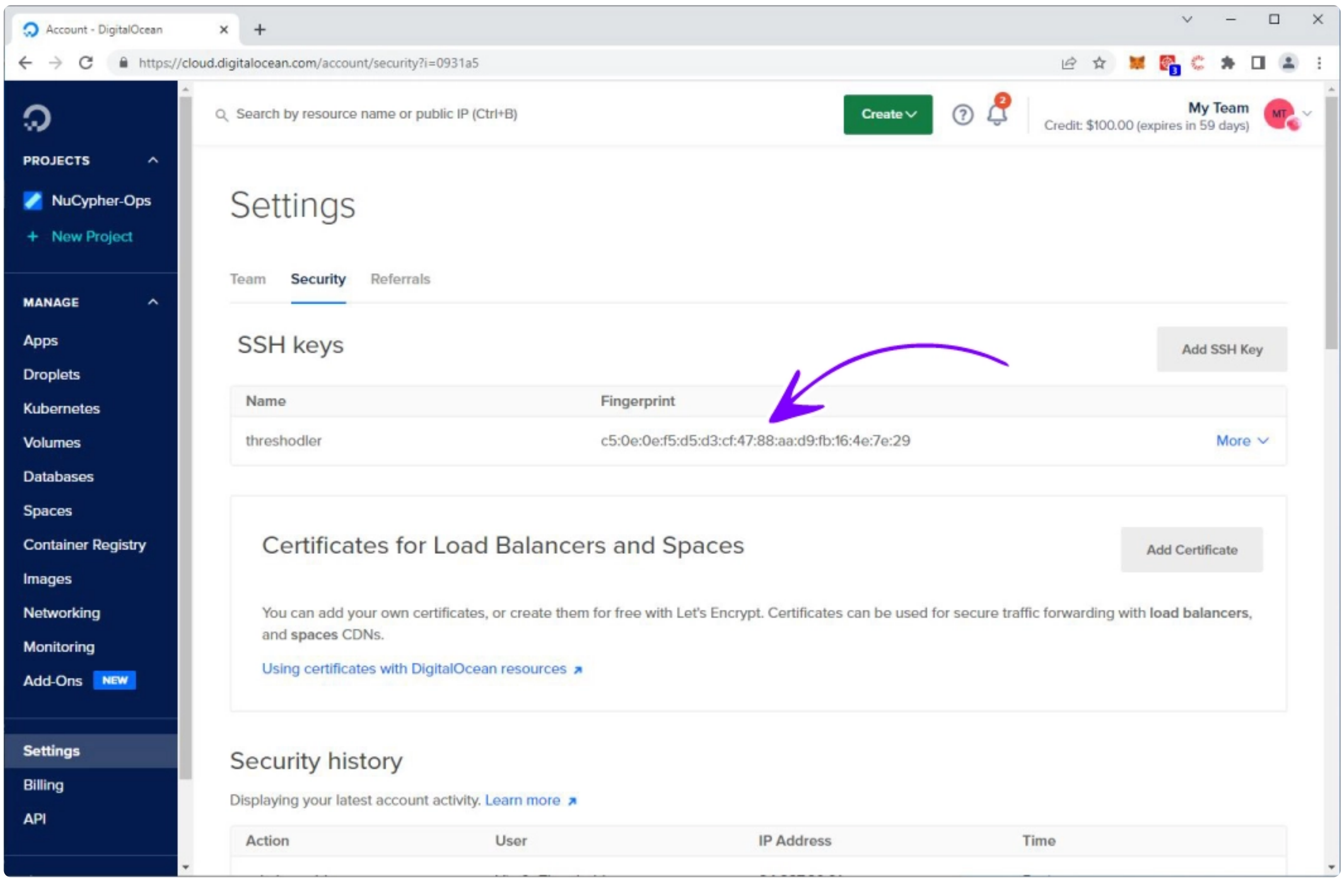
You will be prompted to save and name the key.

```
Generating public/private rsa key
pair. Enter file in which to save
the key (/Users/USER/.ssh/id_rsa):
```

Adding a new SSH Key

Open a Text Document to make notes. Copy the SSH Fingerprint into this text file.

> ⓘ Do not share the fingerprint with anyone. Save and store in a secure place such as a password manager.



Settings

SSH keys

Name	Fingerprint
threshodler	c5:0e:0e:f5:d5:d3:cf:47:88:aa:d9:fb:16:4e:7e:29

Certificates for Load Balancers and Spaces

You can add your own certificates, or create them for free with Let's Encrypt. Certificates can be used for secure traffic forwarding with **load balancers**, and **spaces** CDNs.

Using certificates with DigitalOcean resources

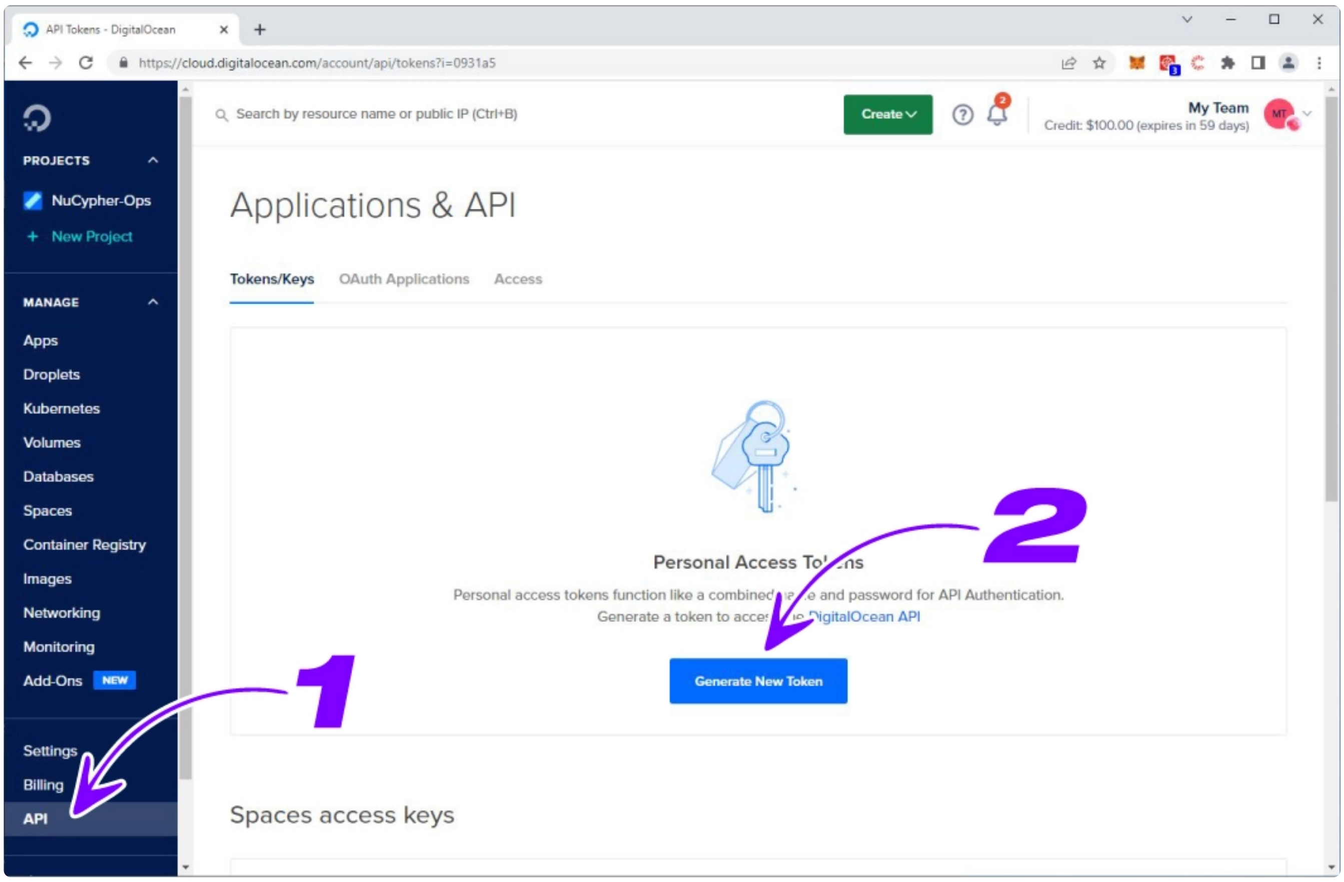
Security history

Displaying your latest account activity. Learn more

Action	User	IP Address	Time

SSH Fingerprint

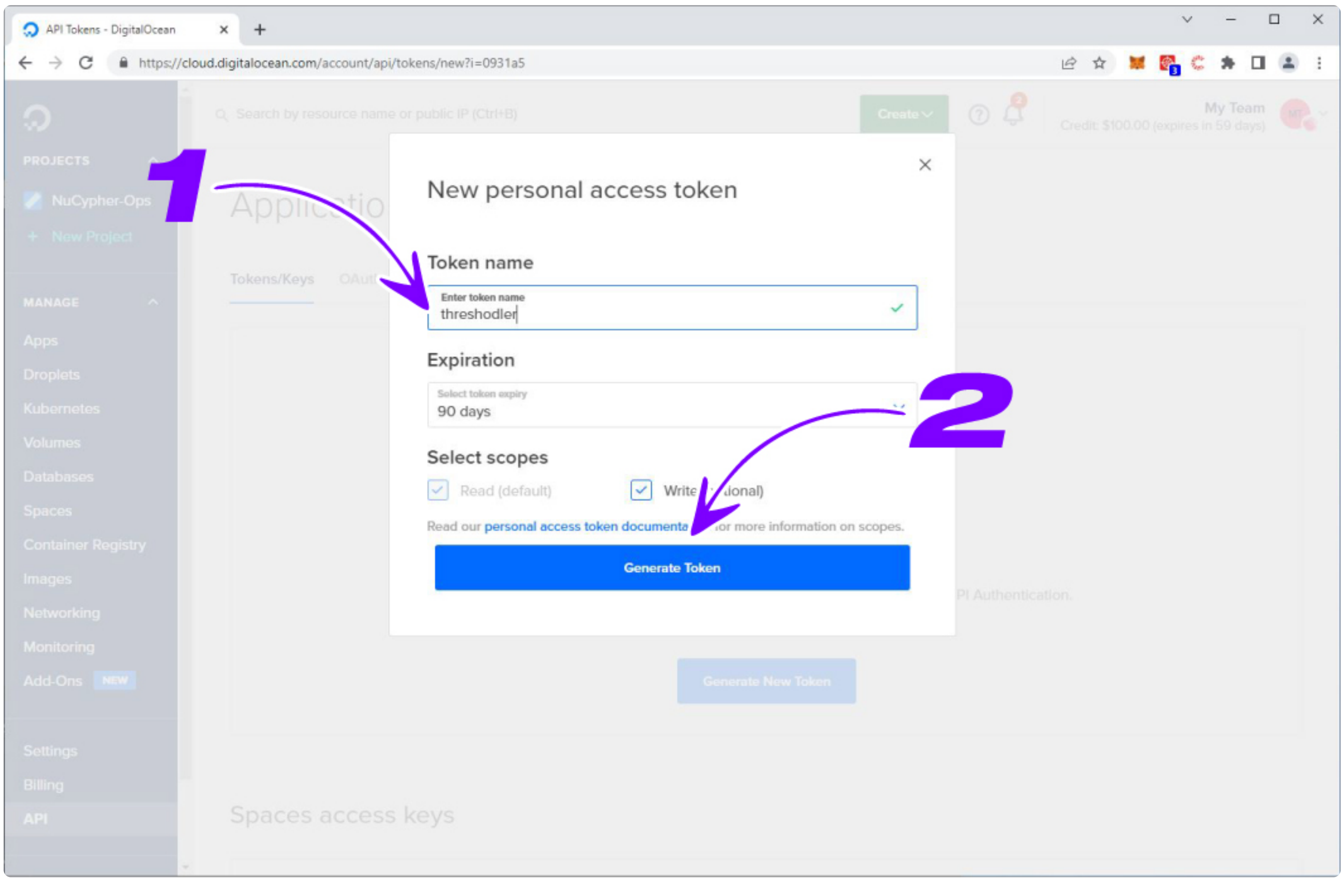
While we're here, let's create an API key as well. Go to **API** > **Tokens/Keys** and click the **Generate New Token** button.


API Tokens - DigitalOcean
https://cloud.digitalocean.com/account/api/tokens?i=0931a5
Search by resource name or public IP (Ctrl+B)
Create
My Team
Credit: $100.00 (expires in 59 days)
PROJECTS
NuCypher-Ops
New Project
MANAGE
Apps
Droplets
Kubernetes
Volumes
Databases
Spaces
Container Registry
Images
Networking
Monitoring
Add-Ons NEW
Settings
Billing
API
Applications & API
Tokens/Keys
OAuth Applications
Access
Personal Access Tokens
Personal access tokens function like a combined name and password for API Authentication.
Generate a token to access the DigitalOcean API
Generate New Token
Spaces access keys
1
2

Generating an API Access Token

Name your token and click the **Generate New Token** button.

ⓘ Unless you plan to periodically refresh this token yourself, you can set "Expiration" to NEVER



New personal access token

Token name

Enter token name: threshodler

Expiration

Select token expiry: 90 days

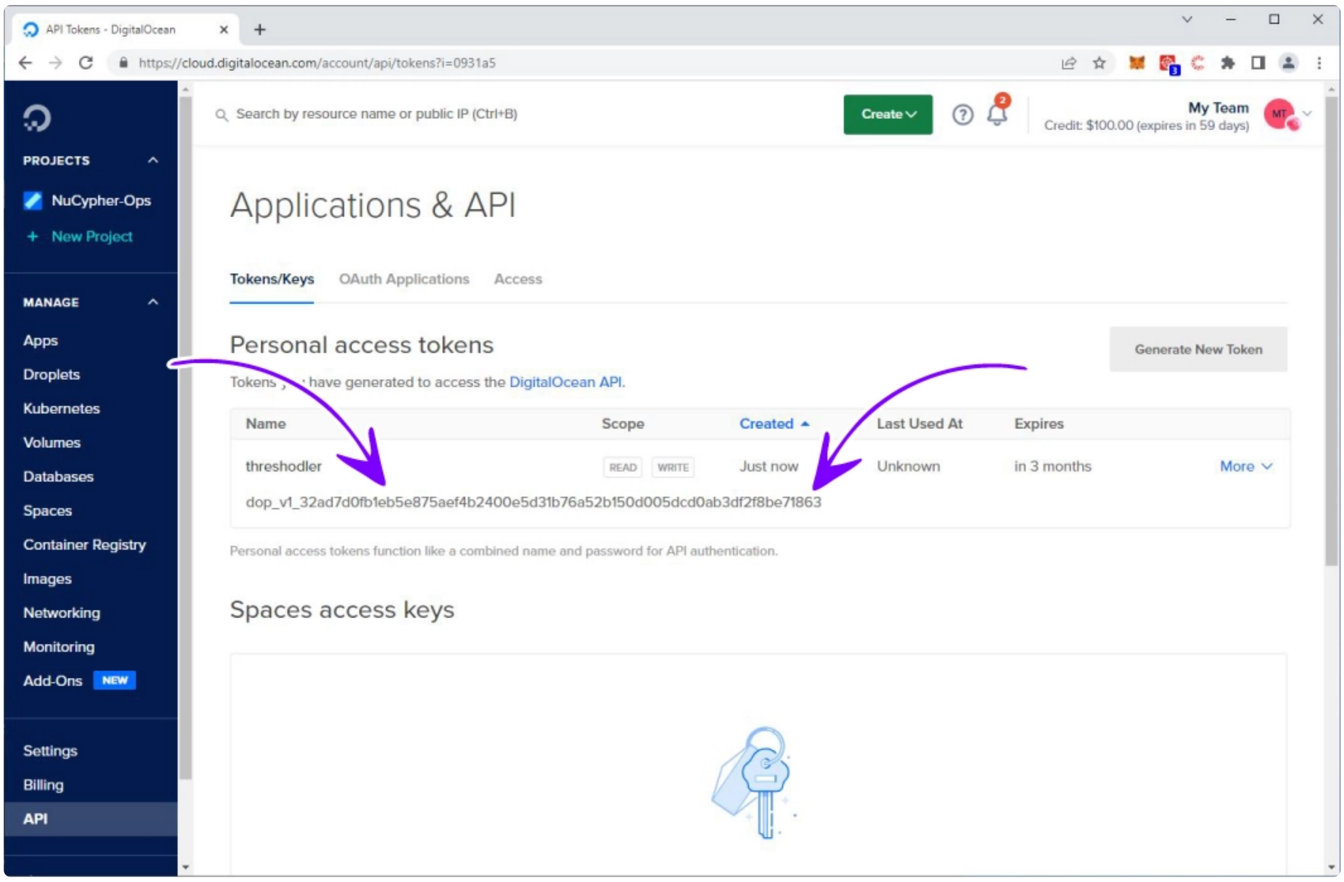
Select scopes

✓ Read (default) ✓ Write (optional)

Read our personal access token documentation for more information on scopes.

Generating an API Access Token

Copy the API Key into your Text file or password manager.



Applications & API

Personal access tokens

Tokens you have generated to access the DigitalOcean API.

Name	Scope	Created	Last Used At	Expires
threshodler	READ WRITE	Just now	Unknown	in 3 months

dop_v1_32ad7d0fb1eb5e875aef4b2400e5d31b76a52b150d005dcd0ab3df2f8be71863

Personal access tokens function like a combined name and password for API authentication.

Spaces access keys

Generating an API Access Token

L2 Providers

The PRE application requires access to an Ethereum endpoint, as well as a Polygon endpoint. One solution is to use a provider such as Infura, Quicknode, or Alchemy. Each has their own pros and cons. Review them carefully.

Once you have made your decision(s) and signed up for an account, login to your account and setup an endpoint for Ethereum. The setup steps are very straightforward for each provider.

Make sure you select **Mainnet** for each endpoint. You will receive what looks like a https web address, Infura calls it an Endpoint, Quicknode refers to it as a Web3 endpoint. Make sure that the endpoint you copy starts with https.

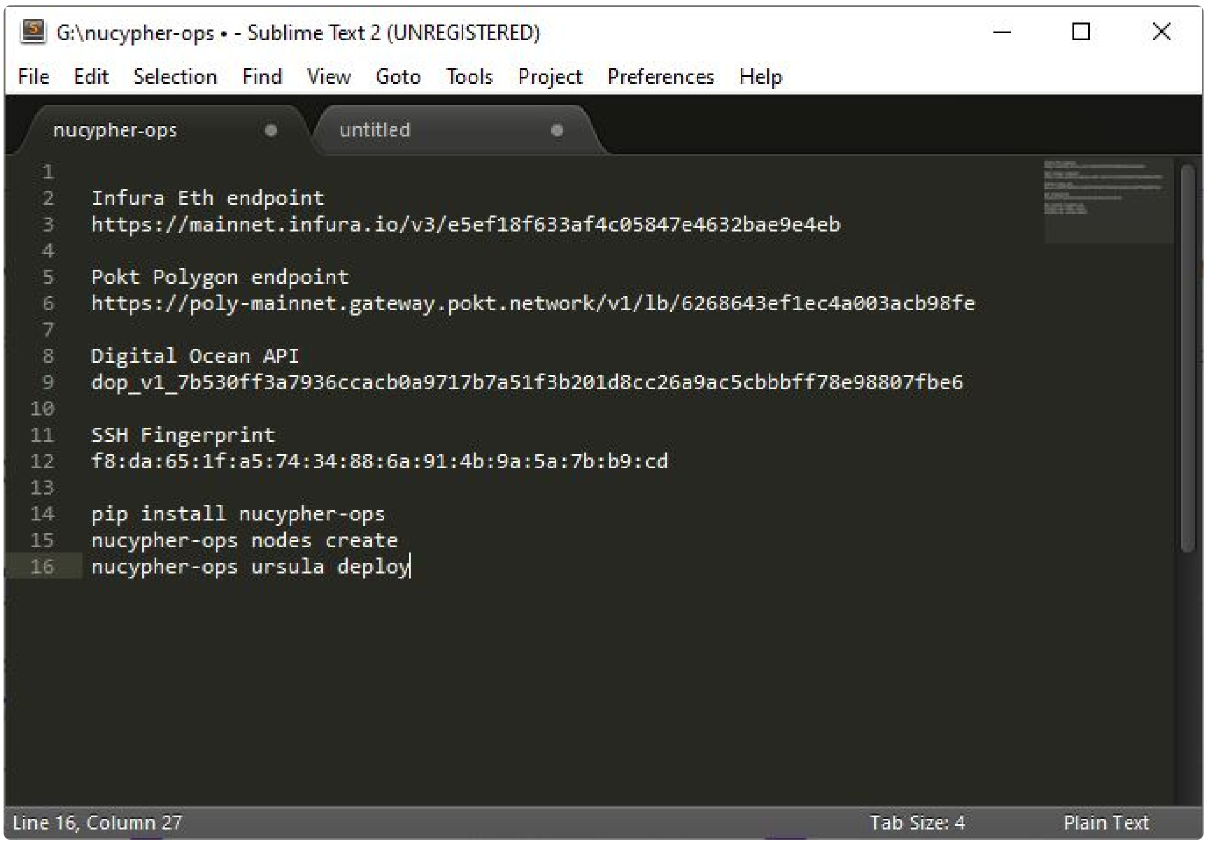
Copy each of the endpoints to your text document or into your password manager. We will use them shortly.

> Do NOT share your provider addresses with anyone.

Checklist

Let's review and make sure you have everything needed to be successful:

- SSH key pair fingerprint
- API Access token
- Endpoint URL for Ethereum
- Endpoint URL for Polygon


G:\nucypher-ops • - Sublime Text 2 (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
nucypher-ops
untitled
Line 16, Column 27
Tab Size: 4
Plain Text

```

Infura Eth endpoint
https://mainnet.infura.io/v3/e5ef18f633af4c05847e4632bae9e4eb

Pokt Polygon endpoint
https://poly-mainnet.gateway.pokt.network/v1/lb/6268643ef1ec4a003acb98fe

Digital Ocean API
dop_v1_7b530ff3a7936ccacb0a9717b7a51f3b201d8cc26a9ac5cbbbff78e98807fbe6

SSH Fingerprint
f8:da:65:1f:a5:74:34:88:6a:91:4b:9a:5a:7b:b9:cd

pip install nucypher-ops
nucypher-ops nodes create
nucypher-ops ursula deploy
```

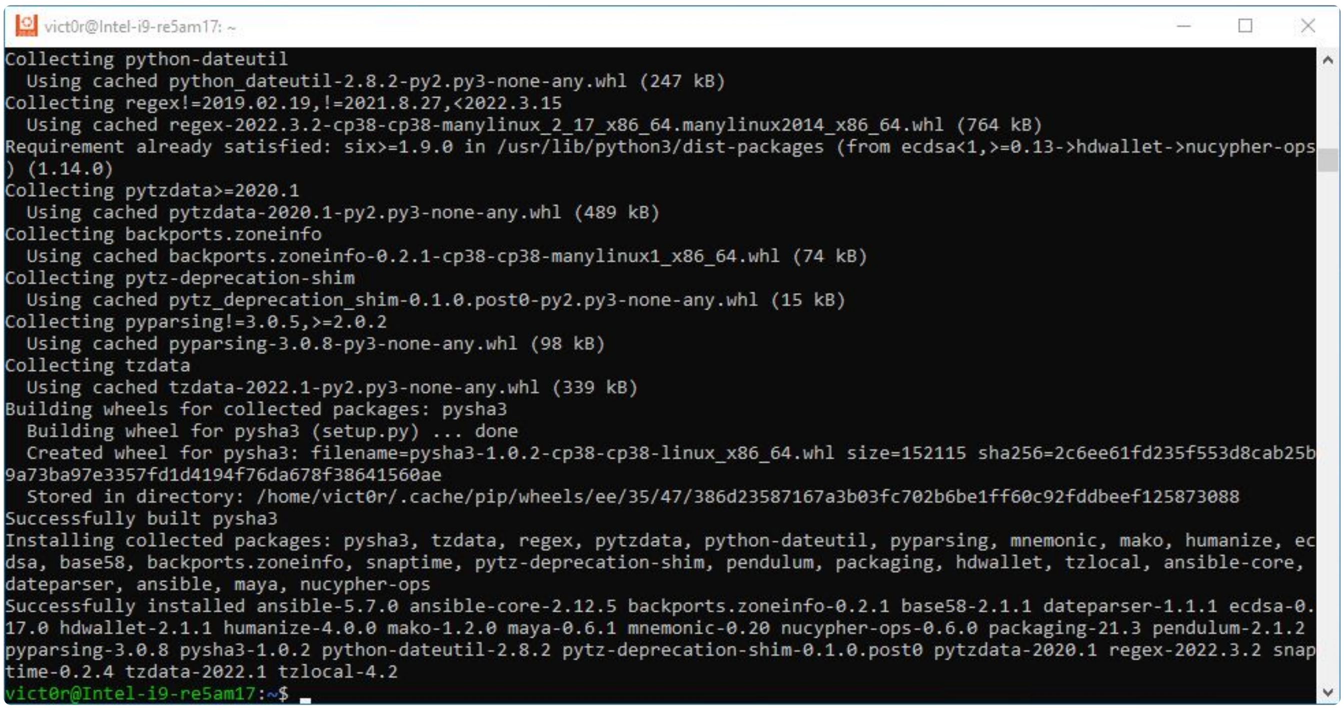
Sample text file

These items should be kept readily available.

Installing `nucypher-ops`

```
pip install nucypher-ops
```

This will download the utility, and install dependencies. This process will take several minutes.



```
vict0r@Intel-i9-re5am17: ~
Collecting python-dateutil
  Using cached python_dateutil-2.8.2-py2.py3-none-any.whl (247 kB)
Collecting regex!=2019.02.19,!=2021.8.27,<2022.3.15
  Using cached regex-2022.3.2-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (764 kB)
Requirement already satisfied: six>=1.9.0 in /usr/lib/python3/dist-packages (from ecdsa<1,>=0.13->hdwallet->nucypher-ops
) (1.14.0)
Collecting pytzdata>=2020.1
  Using cached pytzdata-2020.1-py2.py3-none-any.whl (489 kB)
Collecting backports.zoneinfo
  Using cached backports.zoneinfo-0.2.1-cp38-cp38-manylinux1_x86_64.whl (74 kB)
Collecting pytz-deprecation-shim
  Using cached pytz_deprecation_shim-0.1.0.post0-py2.py3-none-any.whl (15 kB)
Collecting pyparsing!=3.0.5,>=2.0.2
  Using cached pyparsing-3.0.8-py3-none-any.whl (98 kB)
Collecting tzdata
  Using cached tzdata-2022.1-py2.py3-none-any.whl (339 kB)
Building wheels for collected packages: pysha3
  Building wheel for pysha3 (setup.py) ... done
  Created wheel for pysha3: filename=pysha3-1.0.2-cp38-cp38-linux_x86_64.whl size=152115 sha256=2c6ee61fd235f553d8cab25b
9a73ba97e3357fd1d4194f76da678f38641560ae
  Stored in directory: /home/vict0r/.cache/pip/wheels/ee/35/47/386d23587167a3b03fc702b6be1ff60c92fddbeef125873088
Successfully built pysha3
Installing collected packages: pysha3, tzdata, regex, pytzdata, python-dateutil, pyparsing, mnemonic, mako, humanize, ec
dsa, base58, backports.zoneinfo, snaptime, pytz-deprecation-shim, pendulum, packaging, hdwallet, tzlocal, ansible-core,
dateparser, ansible, maya, nucypher-ops
Successfully installed ansible-5.7.0 ansible-core-2.12.5 backports.zoneinfo-0.2.1 base58-2.1.1 dateparser-1.1.1 ecdsa-0.
17.0 hdwallet-2.1.1 humanize-4.0.0 mako-1.2.0 maya-0.6.1 mnemonic-0.20 nucypher-ops-0.6.0 packaging-21.3 pendulum-2.1.2
pyparsing-3.0.8 pysha3-1.0.2 python-dateutil-2.8.2 pytz-deprecation-shim-0.1.0.post0 pytzdata-2020.1 regex-2022.3.2 snap
time-0.2.4 tzdata-2022.1 tzlocal-4.2
vict0r@Intel-i9-re5am17:~$
```

nucypher-ops installation

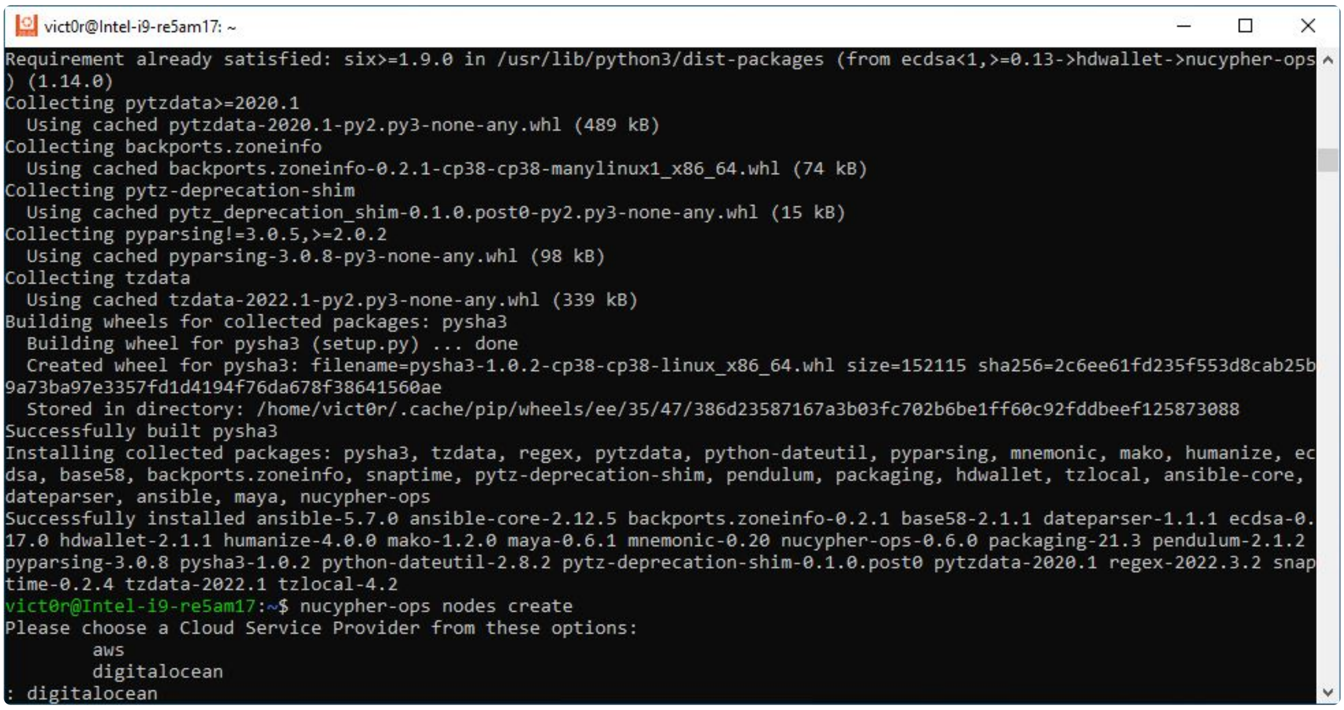
Creating Your Node

Execute the following and be ready to answer questions from the utility:

```
nucypher-ops nodes create
```

This will connect to your Digital Ocean account, spin up a VPS, configure and secure it, and install all of the necessary software for you.

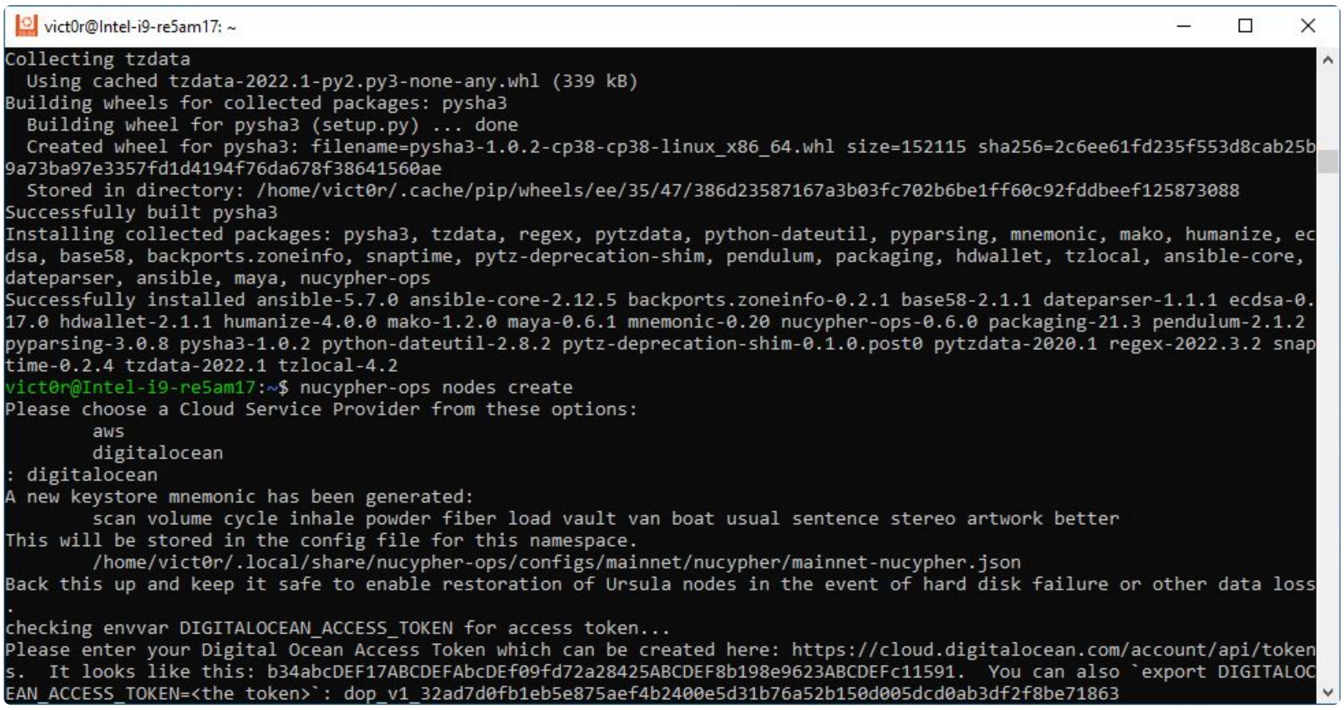
The utility will ask which provider you are using. Answer accordingly and press **ENTER**.



```
vict0r@Intel-i9-re5am17: ~
Requirement already satisfied: six>=1.9.0 in /usr/lib/python3/dist-packages (from ecdsa<1,>=0.13->hdwallet->nucypher-ops
) (1.14.0)
Collecting pytzdata>=2020.1
  Using cached pytzdata-2020.1-py2.py3-none-any.whl (489 kB)
Collecting backports.zoneinfo
  Using cached backports.zoneinfo-0.2.1-cp38-cp38-manylinux1_x86_64.whl (74 kB)
Collecting pytz-deprecation-shim
  Using cached pytz_deprecation_shim-0.1.0.post0-py2.py3-none-any.whl (15 kB)
Collecting pyparsing!=3.0.5,>=2.0.2
  Using cached pyparsing-3.0.8-py3-none-any.whl (98 kB)
Collecting tzdata
  Using cached tzdata-2022.1-py2.py3-none-any.whl (339 kB)
Building wheels for collected packages: pysha3
  Building wheel for pysha3 (setup.py) ... done
  Created wheel for pysha3: filename=pysha3-1.0.2-cp38-cp38-linux_x86_64.whl size=152115 sha256=2c6ee61fd235f553d8cab25b
9a73ba97e3357fd1d4194f76da678f38641560ae
  Stored in directory: /home/vict0r/.cache/pip/wheels/ee/35/47/386d23587167a3b03fc702b6be1ff60c92fddbeef125873088
Successfully built pysha3
Installing collected packages: pysha3, tzdata, regex, pytzdata, python-dateutil, pyparsing, mnemonic, mako, humanize, ec
dsa, base58, backports.zoneinfo, snaptime, pytz-deprecation-shim, pendulum, packaging, hdwallet, tzlocal, ansible-core,
dateparser, ansible, maya, nucypher-ops
Successfully installed ansible-5.7.0 ansible-core-2.12.5 backports.zoneinfo-0.2.1 base58-2.1.1 dateparser-1.1.1 ecdsa-0.
17.0 hdwallet-2.1.1 humanize-4.0.0 mako-1.2.0 maya-0.6.1 mnemonic-0.20 nucypher-ops-0.6.0 packaging-21.3 pendulum-2.1.2
pyparsing-3.0.8 pysha3-1.0.2 python-dateutil-2.8.2 pytz-deprecation-shim-0.1.0.post0 pytzdata-2020.1 regex-2022.3.2 snap
time-0.2.4 tzdata-2022.1 tzlocal-4.2
vict0r@Intel-i9-re5am17:~$ nucypher-ops nodes create
Please choose a Cloud Service Provider from these options:
        aws
        digitalocean
: digitalocean
```

Selecting VPS provider in nucypher-ops

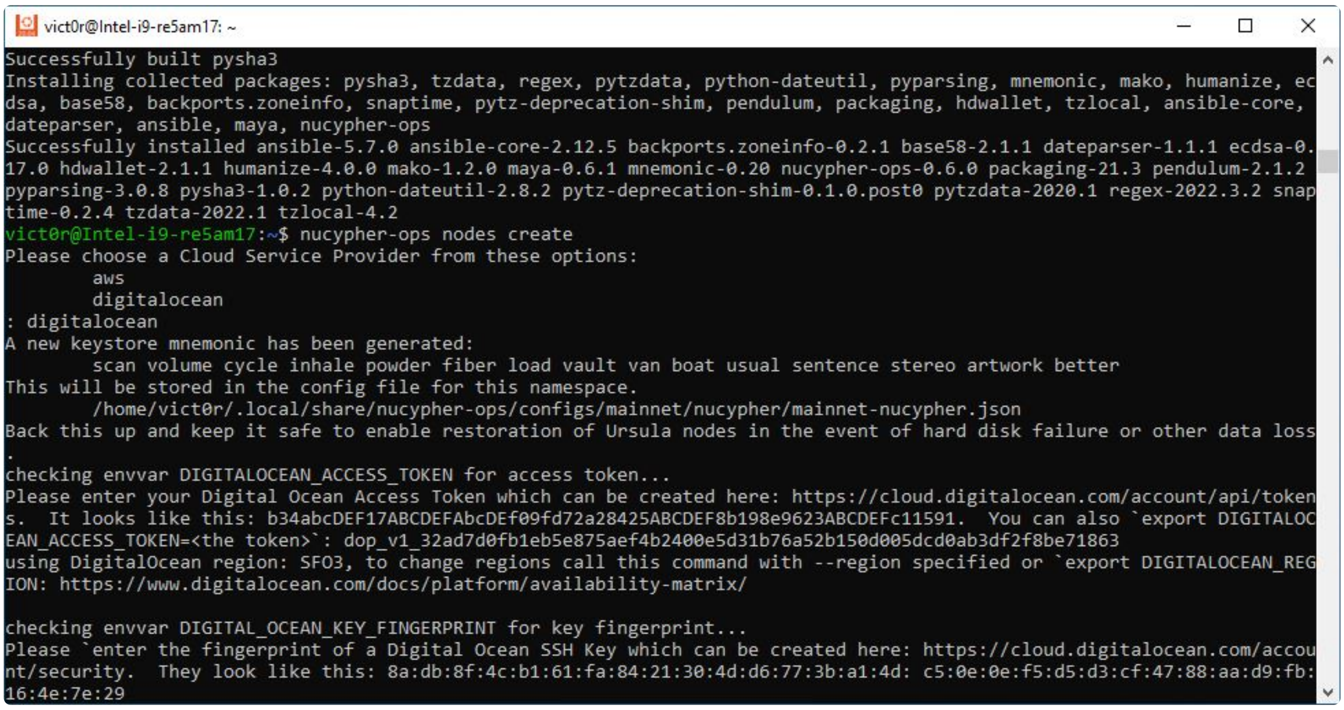
Provide your API Key and press **ENTER**.



```
vict0r@Intel-i9-re5am17: ~
Collecting tzdata
  Using cached tzdata-2022.1-py2.py3-none-any.whl (339 kB)
Building wheels for collected packages: pysha3
  Building wheel for pysha3 (setup.py) ... done
  Created wheel for pysha3: filename=pysha3-1.0.2-cp38-cp38-linux_x86_64.whl size=152115 sha256=2c6ee61fd235f553d8cab25b
9a73ba97e3357fd1d4194f76da678f38641560ae
  Stored in directory: /home/vict0r/.cache/pip/wheels/ee/35/47/386d23587167a3b03fc702b6be1ff60c92fddbeef125873088
Successfully built pysha3
Installing collected packages: pysha3, tzdata, regex, pytzdata, python-dateutil, pyparsing, mnemonic, mako, humanize, ec
dsa, base58, backports.zoneinfo, snaptime, pytz-deprecation-shim, pendulum, packaging, hdwallet, tzlocal, ansible-core,
dateparser, ansible, maya, nucypher-ops
Successfully installed ansible-5.7.0 ansible-core-2.12.5 backports.zoneinfo-0.2.1 base58-2.1.1 dateparser-1.1.1 ecdsa-0.
17.0 hdwallet-2.1.1 humanize-4.0.0 mako-1.2.0 maya-0.6.1 mnemonic-0.20 nucypher-ops-0.6.0 packaging-21.3 pendulum-2.1.2
pyparsing-3.0.8 pysha3-1.0.2 python-dateutil-2.8.2 pytz-deprecation-shim-0.1.0.post0 pytzdata-2020.1 regex-2022.3.2 snap
time-0.2.4 tzdata-2022.1 tzlocal-4.2
vict0r@Intel-i9-re5am17:~$ nucypher-ops nodes create
Please choose a Cloud Service Provider from these options:
        aws
        digitalocean
: digitalocean
A new keystore mnemonic has been generated:
        scan volume cycle inhale powder fiber load vault van boat usual sentence stereo artwork better
This will be stored in the config file for this namespace.
        /home/vict0r/.local/share/nucypher-ops/configs/mainnet/nucypher/mainnet-nucypher.json
Back this up and keep it safe to enable restoration of Ursula nodes in the event of hard disk failure or other data loss
.
checking envvar DIGITALOCEAN_ACCESS_TOKEN for access token...
Please enter your Digital Ocean Access Token which can be created here: https://cloud.digitalocean.com/account/api/token
s.  It looks like this: b34abcDEF17ABCDEFAbcDEf09fd72a28425ABCDEF8b198e9623ABCDEFc11591.  You can also `export DIGITALOC
EAN_ACCESS_TOKEN=<the token>`: dop_v1_32ad7d0fb1eb5e875aef4b2400e5d31b76a52b150d005dcd0ab3df2f8be71863
```

Providing API Key in nucypher-ops

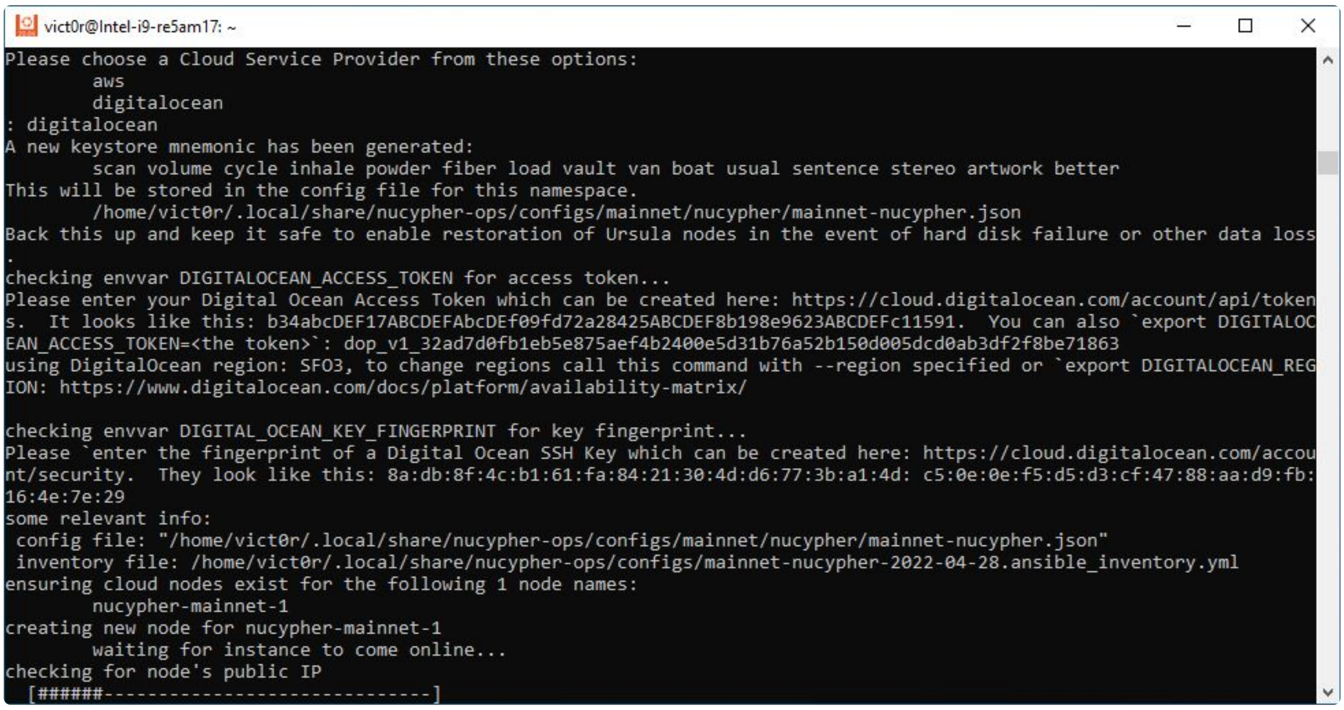
Provide your SSH Fingerprint and press **ENTER**.



```
Successfully built pysha3
Installing collected packages: pysha3, tzdata, regex, pytzdata, python-dateutil, pyparsing, mnemonic, mako, humanize, ec
dsa, base58, backports.zoneinfo, snaptime, pytz-deprecation-shim, pendulum, packaging, hdwallet, tzlocal, ansible-core,
dateparser, ansible, maya, nucypher-ops
Successfully installed ansible-5.7.0 ansible-core-2.12.5 backports.zoneinfo-0.2.1 base58-2.1.1 dateparser-1.1.1 ecdsa-0.
17.0 hdwallet-2.1.1 humanize-4.0.0 mako-1.2.0 maya-0.6.1 mnemonic-0.20 nucypher-ops-0.6.0 packaging-21.3 pendulum-2.1.2
pyparsing-3.0.8 pysha3-1.0.2 python-dateutil-2.8.2 pytz-deprecation-shim-0.1.0.post0 pytzdata-2020.1 regex-2022.3.2 snap
time-0.2.4 tzdata-2022.1 tzlocal-4.2
vict0r@Intel-i9-re5am17:~$ nucypher-ops nodes create
Please choose a Cloud Service Provider from these options:
        aws
        digitalocean
: digitalocean
A new keystore mnemonic has been generated:
        scan volume cycle inhale powder fiber load vault van boat usual sentence stereo artwork better
This will be stored in the config file for this namespace.
        /home/vict0r/.local/share/nucypher-ops/configs/mainnet/nucypher/mainnet-nucypher.json
Back this up and keep it safe to enable restoration of Ursula nodes in the event of hard disk failure or other data loss
.
checking envvar DIGITALOCEAN_ACCESS_TOKEN for access token...
Please enter your Digital Ocean Access Token which can be created here: https://cloud.digitalocean.com/account/api/token
s.  It looks like this: b34abcDEF17ABCDEFAbcDEf09fd72a28425ABCDEF8b198e9623ABCDEFc11591.  You can also `export DIGITALOC
EAN_ACCESS_TOKEN=<the token>`: dop_v1_32ad7d0fb1eb5e875aef4b2400e5d31b76a52b150d005dcd0ab3df2f8be71863
using DigitalOcean region: SFO3, to change regions call this command with --region specified or `export DIGITALOCEAN_REG
ION: https://www.digitalocean.com/docs/platform/availability-matrix/

checking envvar DIGITAL_OCEAN_KEY_FINGERPRINT for key fingerprint...
Please `enter the fingerprint of a Digital Ocean SSH Key which can be created here: https://cloud.digitalocean.com/accou
nt/security.  They look like this: 8a:db:8f:4c:b1:61:fa:84:21:30:4d:d6:77:3b:a1:4d: c5:0e:0e:f5:d5:d3:cf:47:88:aa:d9:fb:
16:4e:7e:29
```

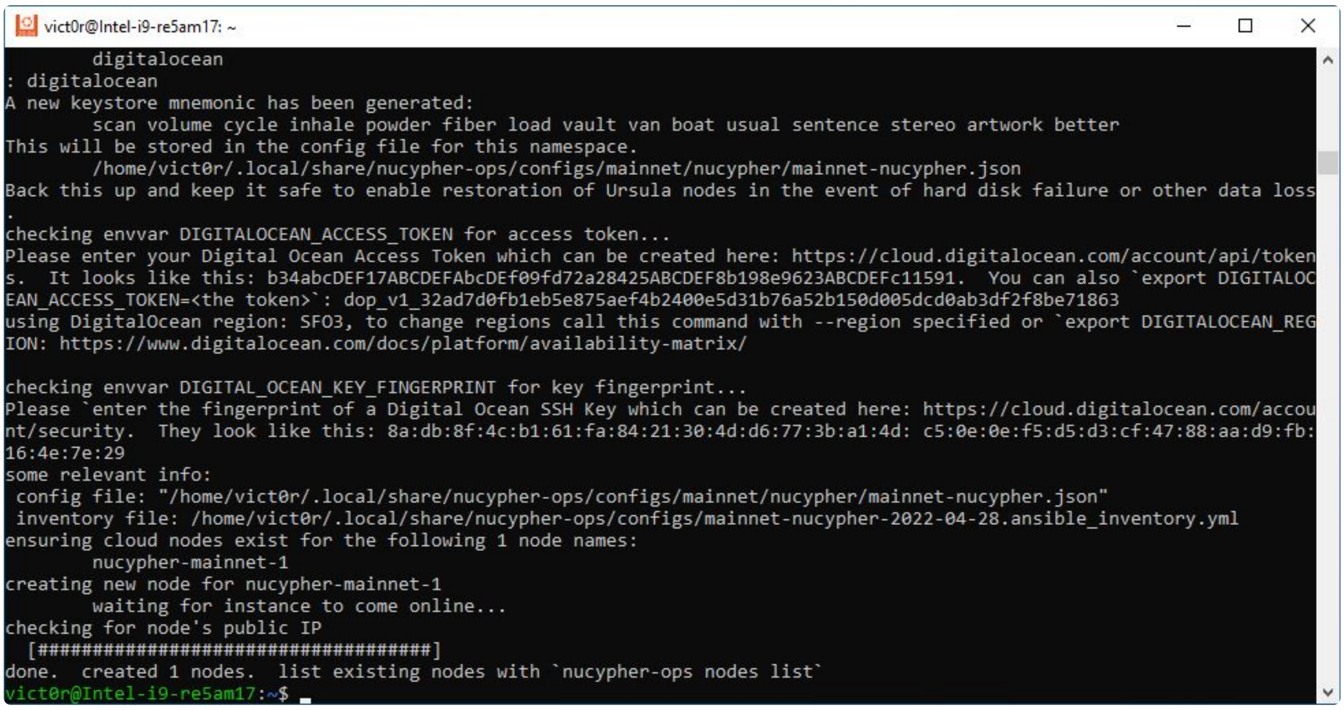
Providing SSH fingerprint in nucypher-ops

Then be patient.



```
Please choose a Cloud Service Provider from these options:
        aws
        digitalocean
: digitalocean
A new keystore mnemonic has been generated:
        scan volume cycle inhale powder fiber load vault van boat usual sentence stereo artwork better
This will be stored in the config file for this namespace.
        /home/vict0r/.local/share/nucypher-ops/configs/mainnet/nucypher/mainnet-nucypher.json
Back this up and keep it safe to enable restoration of Ursula nodes in the event of hard disk failure or other data loss
.
checking envvar DIGITALOCEAN_ACCESS_TOKEN for access token...
Please enter your Digital Ocean Access Token which can be created here: https://cloud.digitalocean.com/account/api/token
s.  It looks like this: b34abcDEF17ABCDEFAbcDEf09fd72a28425ABCDEF8b198e9623ABCDEFc11591.  You can also `export DIGITALOC
EAN_ACCESS_TOKEN=<the token>`: dop_v1_32ad7d0fb1eb5e875aef4b2400e5d31b76a52b150d005dcd0ab3df2f8be71863
using DigitalOcean region: SFO3, to change regions call this command with --region specified or `export DIGITALOCEAN_REG
ION: https://www.digitalocean.com/docs/platform/availability-matrix/

checking envvar DIGITAL_OCEAN_KEY_FINGERPRINT for key fingerprint...
Please `enter the fingerprint of a Digital Ocean SSH Key which can be created here: https://cloud.digitalocean.com/accou
nt/security.  They look like this: 8a:db:8f:4c:b1:61:fa:84:21:30:4d:d6:77:3b:a1:4d: c5:0e:0e:f5:d5:d3:cf:47:88:aa:d9:fb:
16:4e:7e:29
some relevant info:
 config file: "/home/vict0r/.local/share/nucypher-ops/configs/mainnet/nucypher/mainnet-nucypher.json"
 inventory file: /home/vict0r/.local/share/nucypher-ops/configs/mainnet-nucypher-2022-04-28.ansible_inventory.yml
ensuring cloud nodes exist for the following 1 node names:
        nucypher-mainnet-1
creating new node for nucypher-mainnet-1
        waiting for instance to come online...
checking for node's public IP
  [######-----------------------------]
```

Creating VPS instance & PRE node



```
        digitalocean
: digitalocean
A new keystore mnemonic has been generated:
        scan volume cycle inhale powder fiber load vault van boat usual sentence stereo artwork better
This will be stored in the config file for this namespace.
        /home/vict0r/.local/share/nucypher-ops/configs/mainnet/nucypher/mainnet-nucypher.json
Back this up and keep it safe to enable restoration of Ursula nodes in the event of hard disk failure or other data loss
.
checking envvar DIGITALOCEAN_ACCESS_TOKEN for access token...
Please enter your Digital Ocean Access Token which can be created here: https://cloud.digitalocean.com/account/api/token
s.  It looks like this: b34abcDEF17ABCDEFAbcDEf09fd72a28425ABCDEF8b198e9623ABCDEFc11591.  You can also `export DIGITALOC
EAN_ACCESS_TOKEN=<the token>`: dop_v1_32ad7d0fb1eb5e875aef4b2400e5d31b76a52b150d005dcd0ab3df2f8be71863
using DigitalOcean region: SFO3, to change regions call this command with --region specified or `export DIGITALOCEAN_REG
ION: https://www.digitalocean.com/docs/platform/availability-matrix/

checking envvar DIGITAL_OCEAN_KEY_FINGERPRINT for key fingerprint...
Please `enter the fingerprint of a Digital Ocean SSH Key which can be created here: https://cloud.digitalocean.com/accou
nt/security.  They look like this: 8a:db:8f:4c:b1:61:fa:84:21:30:4d:d6:77:3b:a1:4d: c5:0e:0e:f5:d5:d3:cf:47:88:aa:d9:fb:
16:4e:7e:29
some relevant info:
 config file: "/home/vict0r/.local/share/nucypher-ops/configs/mainnet/nucypher/mainnet-nucypher.json"
 inventory file: /home/vict0r/.local/share/nucypher-ops/configs/mainnet-nucypher-2022-04-28.ansible_inventory.yml
ensuring cloud nodes exist for the following 1 node names:
        nucypher-mainnet-1
creating new node for nucypher-mainnet-1
        waiting for instance to come online...
checking for node's public IP
  [###################################]
done.  created 1 nodes.  list existing nodes with `nucypher-ops nodes list`
vict0r@Intel-i9-re5am17:~$
```

VPS instance & PRE node created

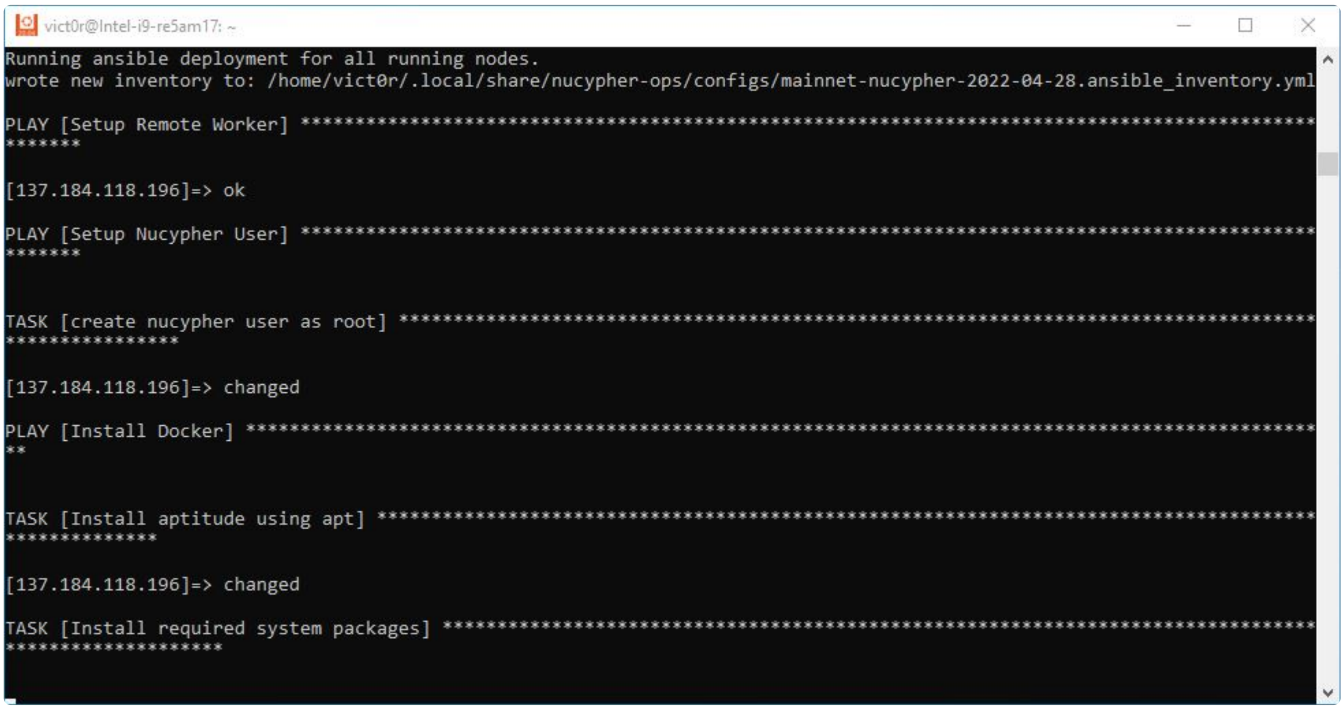
Deploying Ursula

In a few minutes, you'll be ready to bond your stake.

```
nucypher-ops ursula deploy
```

Provide your ETH endpoint when requested and press **ENTER**.

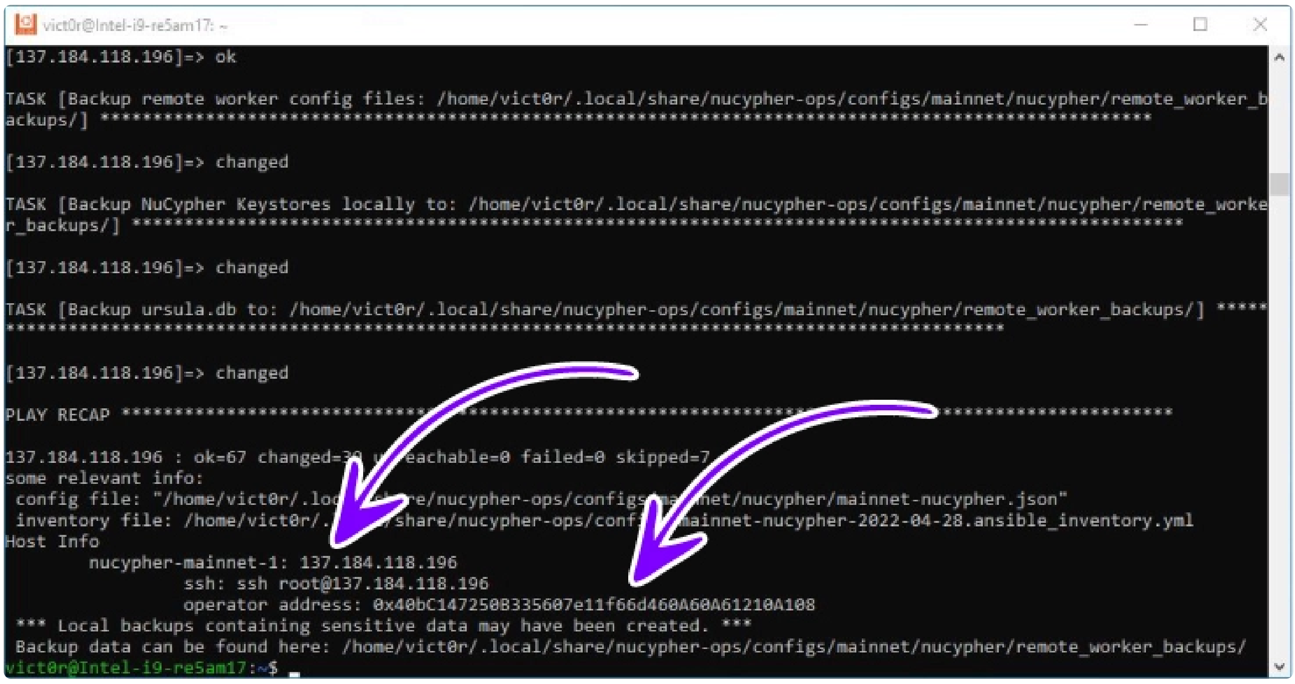
Provide your Polygon endpoint when requested and press **ENTER**.



```
Running ansible deployment for all running nodes.
wrote new inventory to: /home/vict0r/.local/share/nucypher-ops/configs/mainnet-nucypher-2022-04-28.ansible_inventory.yml

PLAY [Setup Remote Worker] *********************************************************************************************
*******

[137.184.118.196]=> ok

PLAY [Setup Nucypher User] *********************************************************************************************
*******


TASK [create nucypher user as root] ************************************************************************************
****************

[137.184.118.196]=> changed

PLAY [Install Docker] **************************************************************************************************
**


TASK [Install aptitude using apt] **************************************************************************************
**************

[137.184.118.196]=> changed

TASK [Install required system packages] ********************************************************************************
********************
```

Configuring and deploying Ursula

Wait for it...



```
[137.184.118.196]=> ok

TASK [Backup remote worker config files: /home/vict0r/.local/share/nucypher-ops/configs/mainnet/nucypher/remote_worker_b
ackups/] ***************************************************************************************************

[137.184.118.196]=> changed

TASK [Backup NuCypher Keystores locally to: /home/vict0r/.local/share/nucypher-ops/configs/mainnet/nucypher/remote_worke
r_backups/] ***************************************************************************************************

[137.184.118.196]=> changed

TASK [Backup ursula.db to: /home/vict0r/.local/share/nucypher-ops/configs/mainnet/nucypher/remote_worker_backups/] *****
***********************************************************************************************

[137.184.118.196]=> changed

PLAY RECAP *************************************************************************************************************

137.184.118.196 : ok=67 changed=30 unreachable=0 failed=0 skipped=7
some relevant info:
 config file: "/home/vict0r/.local/share/nucypher-ops/configs/mainnet/nucypher/mainnet-nucypher.json"
 inventory file: /home/vict0r/.local/share/nucypher-ops/configs/mainnet-nucypher-2022-04-28.ansible_inventory.yml
Host Info
        nucypher-mainnet-1: 137.184.118.196
                 ssh: ssh root@137.184.118.196
                 operator address: 0x40bC147250B335607e11f66d460A60A61210A108
 *** Local backups containing sensitive data may have been created. ***
 Backup data can be found here: /home/vict0r/.local/share/nucypher-ops/configs/mainnet/nucypher/remote_worker_backups/
vict0r@Intel-i9-re5am17:~$
```

PRE node ready to bond

The items at the bottom are your node's IP address and Operator address, to which you will need to bond your stake.

To bond your stake to your new PRE node, click below:

NuCypher Staking (Beta)

Bonding UI link

Make sure you take note of your operator address.

Nucypher-Ops Config File location

Nucypher-ops is a convenient tool for deploying a PRE node on the Threshold network. In case you need to make changes to the configuration of the utility on your local machine, these are the default directories where config files are stored locally:

Linux

```
~/.local/share/nucypher-ops
```

MacOS

```
~/Library/Application Support/nucypher-ops
```

Caution: Do not make changes to these files without making a backup of them first. Do not make changes to these files unless you know and understand what you are doing.

Caution: Breaking these files may lead to your node becoming permanently inaccessible. Use extreme caution.

Post Install

To find out about other features try

```
nucypher-ops --help
```

if you need to update your node to a new version of NuCypher just execute

```
nucypher-ops ursula update
```

Keep in mind staking is an active process, and that you are responsible for operational security, as well as backing up any relevant data. Carefully review your firewall rules and make necessary adjustments. `nucypher-ops` configures a PRE node, it does not setup security measures.

tBTC v2 Node Setup

This document explains the basic installation and configuration for the tBTC v2 staking client.

This document is intended for members of the community who would like to run their own tBTC v2 node.

> Please be aware that running a node is not an easy task and requires technical skill and commitment to maintaining node uptime and availability.

Please review this document in its entirety prior to beginning setup of your node to familiarize yourself with the general setup process.

Important Considerations

The Threshold Network expects certain capabilities for each node running on the network. To help attain these capabilities consider the following criteria:

- It is paramount that tBTC v2 nodes remain available to the Network. We strongly encourage a stable and redundant internet connection.
- Equally important is machine uptime and reliability. A VPS is strongly recommended.
- A connection to a production grade self-hosted or third party Ethereum node deployment.
- Persistent and redundant storage that will survive a VM or container rotation, and a disk failure.
- Each node running on the network requires a unique Ethereum Operator Account. The account has to maintain a positive Ether balance at all times.
- Each node running on the network requires a unique IP address or a unique application port running under the same IP.

Recommended Machine Types

> While it is possible to run the client on a local machine, this is not recommended.

Your operating environment will ultimately dictate what machine type to go with. This is particularly relevant if you’re running a containerized solution where multiple applications are sharing VM resources. The below types are sufficient for running one instance of the tBTC v2 Node.

The preferred OS is Ubuntu.

VPS Provider	VPS Type
AWS	c5.large
Azure	F2s v2
Google Cloud	n2-highcpu-2
Self-hosted	2 vCPU / 2 GB RAM / 1 GiB Persistent Storage

Ethereum API

A Keep Node requires a connection to a WebSocket Ethereum API. You should obtain a WS API URL from a service provider (e.g. Alchemy, Infura, Ankr) or run your own Ethereum node (e.g. Geth).

Configuration

The client expects configuration options to be passed as CLI flags or specified in a config file. If you specify an option by using a parameter on the command line, it will override the value read from the configuration file.

Operator Account

Prepare your machine to install the tBTC v2 staking client

The client requires an Ethereum Key File of an Operator Account to connect to the Ethereum chain. This account is created in a subsequent step using Geth (GoEthereum).

The Ethereum Key File is expected to be encrypted with a password. The password has to be provided in a prompt after the client starts or configured as a `KEEP_ETHEREUM_PASSWORD` environment variable.

The Operator Account has to maintain a positive Ether balance at all times. We strongly advise you monitor the account and top-up when its balance gets below 0,5 Ether.

> Please do NOT reuse an operator account that is being used for PRE or other applications.

Install Geth (GoEthereum)

To create a new Ethereum account, install Geth (GoEthereum) and create a new account using the command below. This account will subsequently be referred to as the Operator Account.

```
geth account new --keystore ./operator-key
```

When prompted, provide a password to protect the operator key file.

> Use a password manager to generate a strong password and store it safely. It will be needed again during setup.

> Avoid passwords that contain the following characters: ', ", `, $
> These characters may be interpreted as part of the configuration which can lead to undesirable outcomes that may be extremely time intensive to correct.

Once the process completes, your public key will be displayed. Take note of your Operator Account public key.

> DO NOT LOSE THE PASSWORD TO THE OPERATOR ACCOUNT.

Funding your Operator Account

Your Operator Account will need to maintain a positive ETH balance at all times to ensure proper operation and availability of your tBTC v2 node.

Application Authorization & Operator Registration

The following demonstrates how to authorize applications for tBTC v2 and how to register your operator address via the Threshold Dashboard.

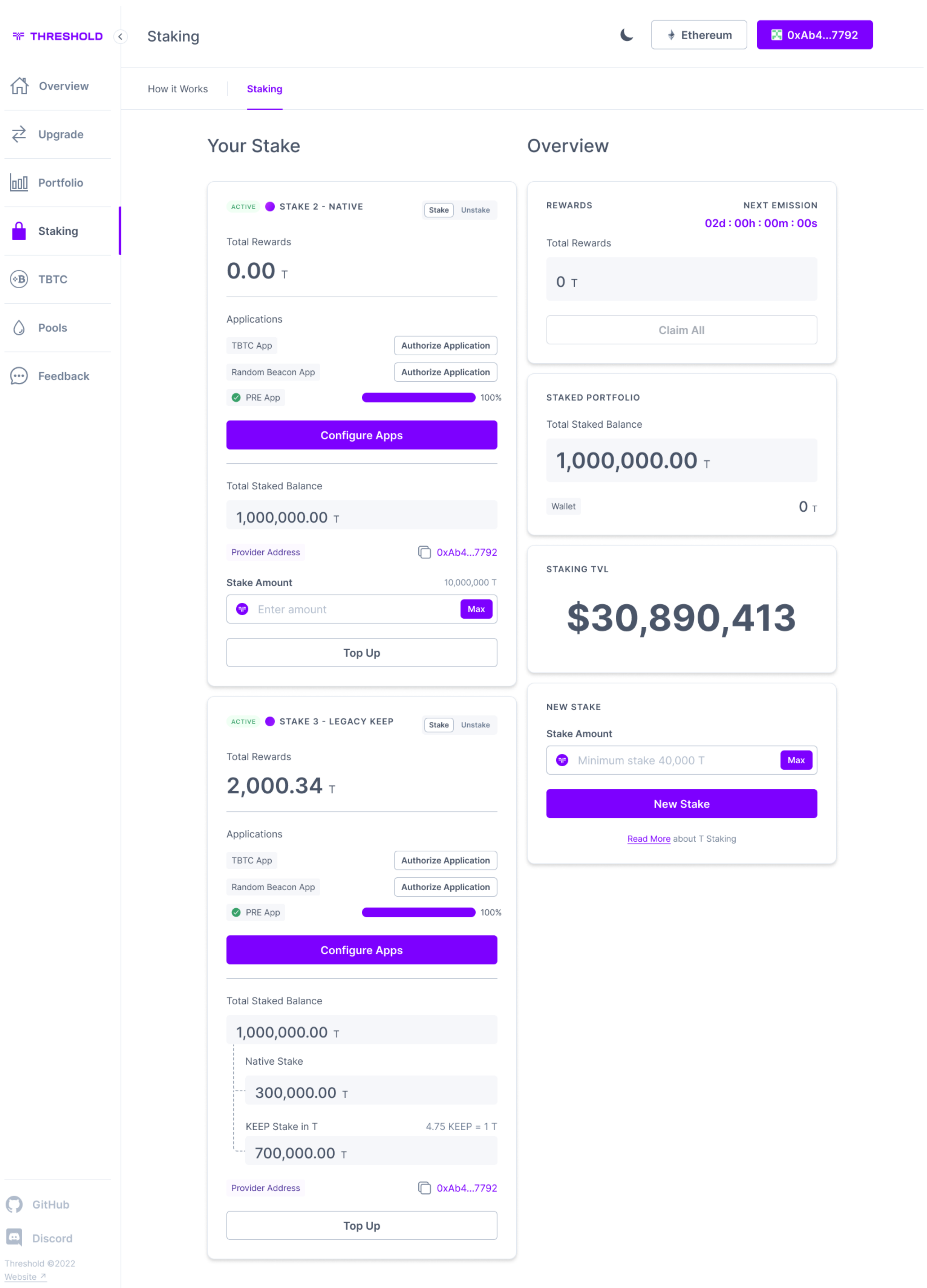
One important step to get your node operating on the Threshold Network is proper application authorization as well as operator account registration. Applications need only be authorized once.

> It is **CRITICALLY** important that both the tBTC application as well as the Random Beacon applications are authorized. A node cannot be deployed without both applications being properly authorized.

> **Please note:** by authorizing these applications, an unbonding period of 45 days will go into effect on the T you stake for these applications. This cool-down period begins the day you submit an unstake request.

Application Authorization

To get started, visit the Threshold Dashboard and connect your wallet.



THRESHOLD

Staking

Ethereum

0xAb4...7792

Overview
Upgrade
Portfolio
Staking
TBTC
Pools
Feedback

How it Works | Staking

Your Stake

ACTIVE ● STAKE 2 - NATIVE | Stake | Unstake

Total Rewards
0.00 T

Applications

TBTC App	Authorize Application
Random Beacon App	Authorize Application
✓ PRE App	100%

Configure Apps

Total Staked Balance
1,000,000.00 T

Provider Address: 0xAb4...7792

Stake Amount: 10,000,000 T
Enter amount | Max

Top Up

ACTIVE ● STAKE 3 - LEGACY KEEP | Stake | Unstake

Total Rewards
2,000.34 T

Applications

TBTC App	Authorize Application
Random Beacon App	Authorize Application
✓ PRE App	100%

Configure Apps

Total Staked Balance
1,000,000.00 T

Native Stake
300,000.00 T

KEEP Stake in T: 4.75 KEEP = 1 T
700,000.00 T

Provider Address: 0xAb4...7792

Top Up

Overview

REWARDS | NEXT EMISSION 02d : 00h : 00m : 00s

Total Rewards
0 T

Claim All

STAKED PORTFOLIO

Total Staked Balance
1,000,000.00 T

Wallet: 0 T

STAKING TVL
$30,890,413

NEW STAKE

Stake Amount
Minimum stake 40,000 T | Max

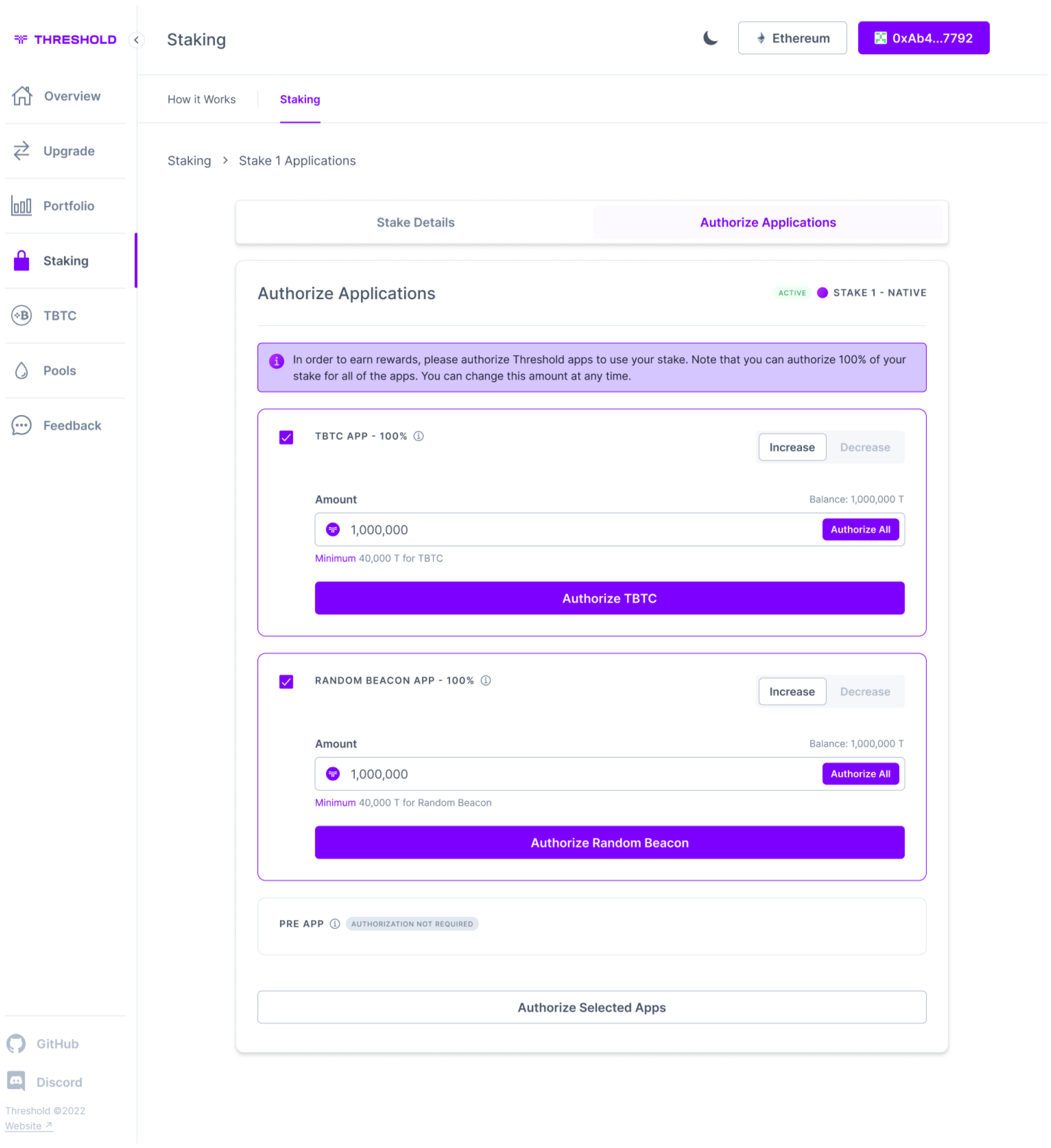
New Stake

Read More about T Staking

GitHub
Discord
Threshold ©2022
Website ↗

Threshold Dashboard

Click on "Configure Apps"



THRESHOLD

Staking

Ethereum

0xAb4...7792

Overview

Upgrade

Portfolio

Staking

TBTC

Pools

Feedback

GitHub

Discord

Threshold ©2022
Website ↗

How it Works | Staking

Staking > Stake 1 Applications

Stake Details | Authorize Applications

Authorize Applications — ACTIVE ● STAKE 1 - NATIVE

In order to earn rewards, please authorize Threshold apps to use your stake. Note that you can authorize 100% of your stake for all of the apps. You can change this amount at any time.

✓ TBTC APP - 100%

Increase | Decrease

Amount — Balance: 1,000,000 T

1,000,000 — Authorize All

Minimum 40,000 T for TBTC

Authorize TBTC

✓ RANDOM BEACON APP - 100%

Increase | Decrease

Amount — Balance: 1,000,000 T

1,000,000 — Authorize All

Minimum 40,000 T for Random Beacon

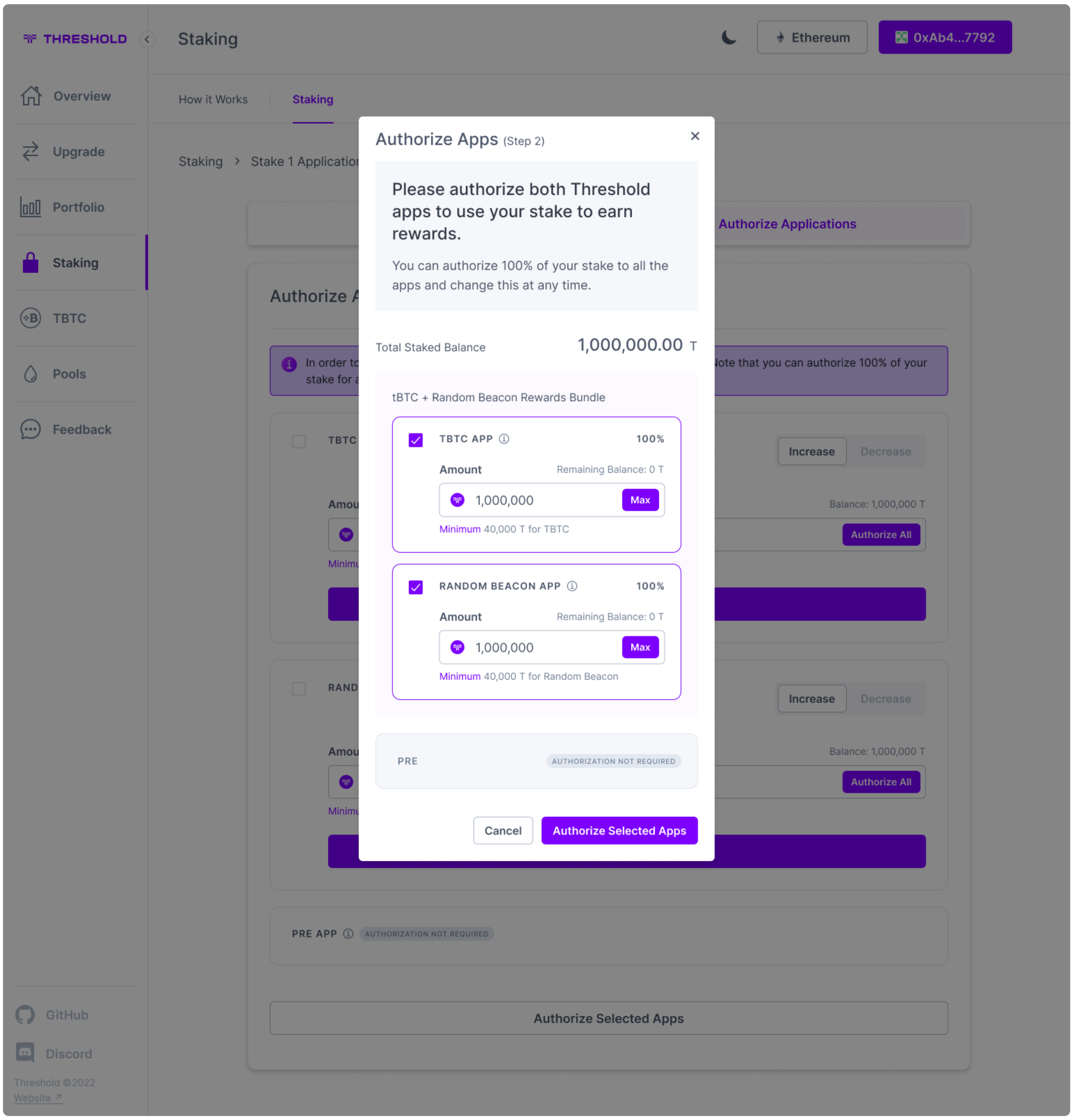
Authorize Random Beacon

PRE APP — AUTHORIZATION NOT REQUIRED

Authorize Selected Apps

Authorizing Threshold Applications

Select BOTH tBTC and Random Beacon applications and enter your desired amount of T to stake per application. Note that the minimum is 40,000T.



Authorize Apps (Step 2)

Please authorize both Threshold apps to use your stake to earn rewards.

You can authorize 100% of your stake to all the apps and change this at any time.

Total Staked Balance 1,000,000.00 T

tBTC + Random Beacon Rewards Bundle

TBTC APP 100%
Amount — Remaining Balance: 0 T
1,000,000 Max
Minimum 40,000 T for TBTC

RANDOM BEACON APP 100%
Amount — Remaining Balance: 0 T
1,000,000 Max
Minimum 40,000 T for Random Beacon

PRE — AUTHORIZATION NOT REQUIRED

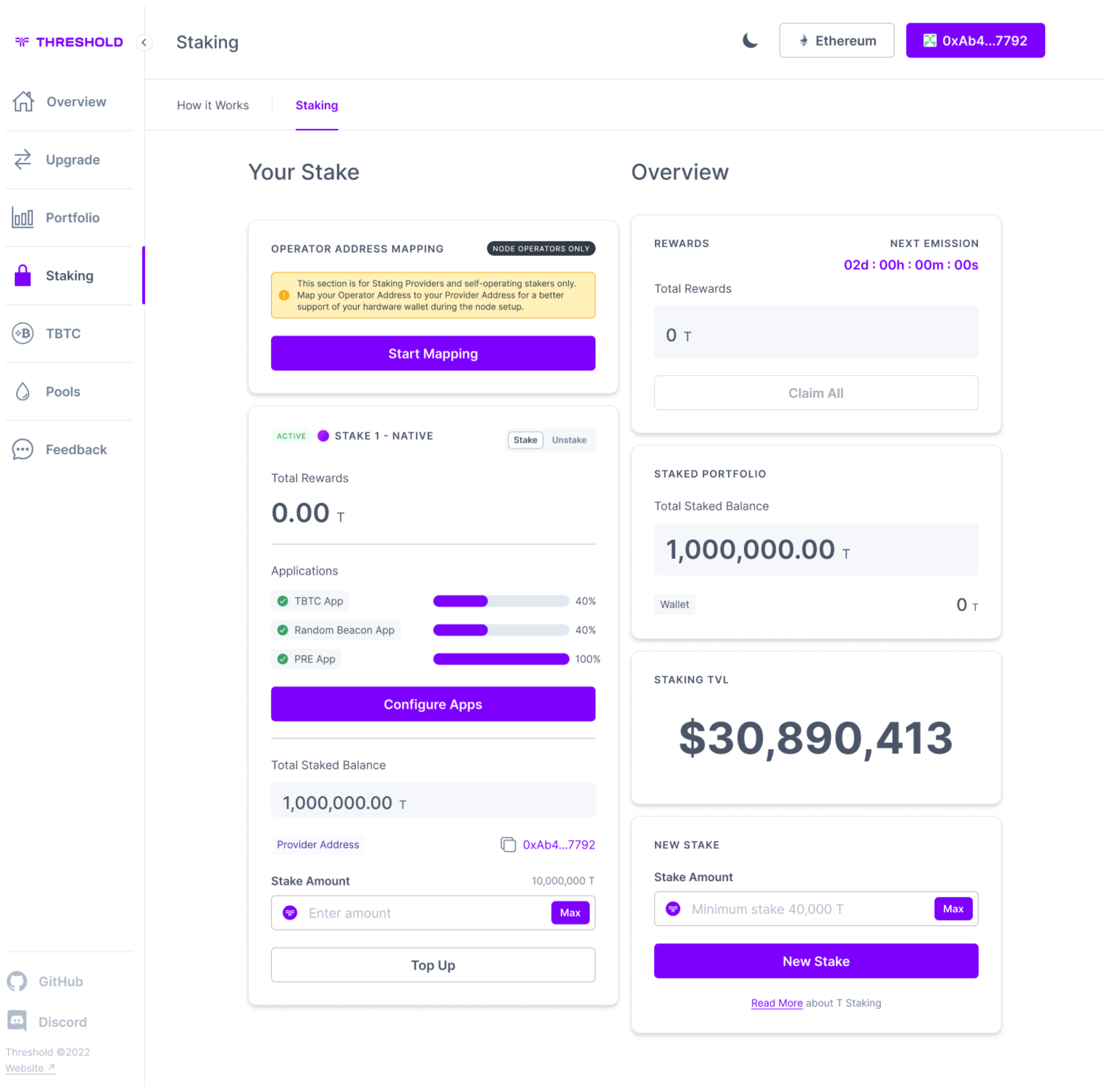
Cancel | Authorize Selected Apps

Authorizing Threshold Applications

Operator Registration

The operator account is the Ethereum account created on your node. In order for the network to associate your T stake with your node, you must register your Operator Address.

> ⓘ An Operator for the Provider registration can be submitted just once. The Operator address assignment cannot be updated.


THRESHOLD

Staking

Ethereum

0xAb4...7792

Overview

Upgrade

Portfolio

Staking

TBTC

Pools

Feedback

GitHub

Discord

Threshold ©2022

Website ↗

How it Works | Staking

Your Stake

OPERATOR ADDRESS MAPPING — NODE OPERATORS ONLY

This section is for Staking Providers and self-operating stakers only. Map your Operator Address to your Provider Address for a better support of your hardware wallet during the node setup.

Start Mapping

ACTIVE ● STAKE 1 - NATIVE — Stake | Unstake

Total Rewards

0.00 T

Applications

TBTC App		40%
Random Beacon App		40%
PRE App		100%

Configure Apps

Total Staked Balance

1,000,000.00 T

Provider Address — 0xAb4...7792

Stake Amount — 10,000,000 T

Enter amount — Max

Top Up

Overview

REWARDS — NEXT EMISSION

02d : 00h : 00m : 00s

Total Rewards

0 T

Claim All

STAKED PORTFOLIO

Total Staked Balance

1,000,000.00 T

Wallet — 0 T

STAKING TVL

$30,890,413

NEW STAKE

Stake Amount

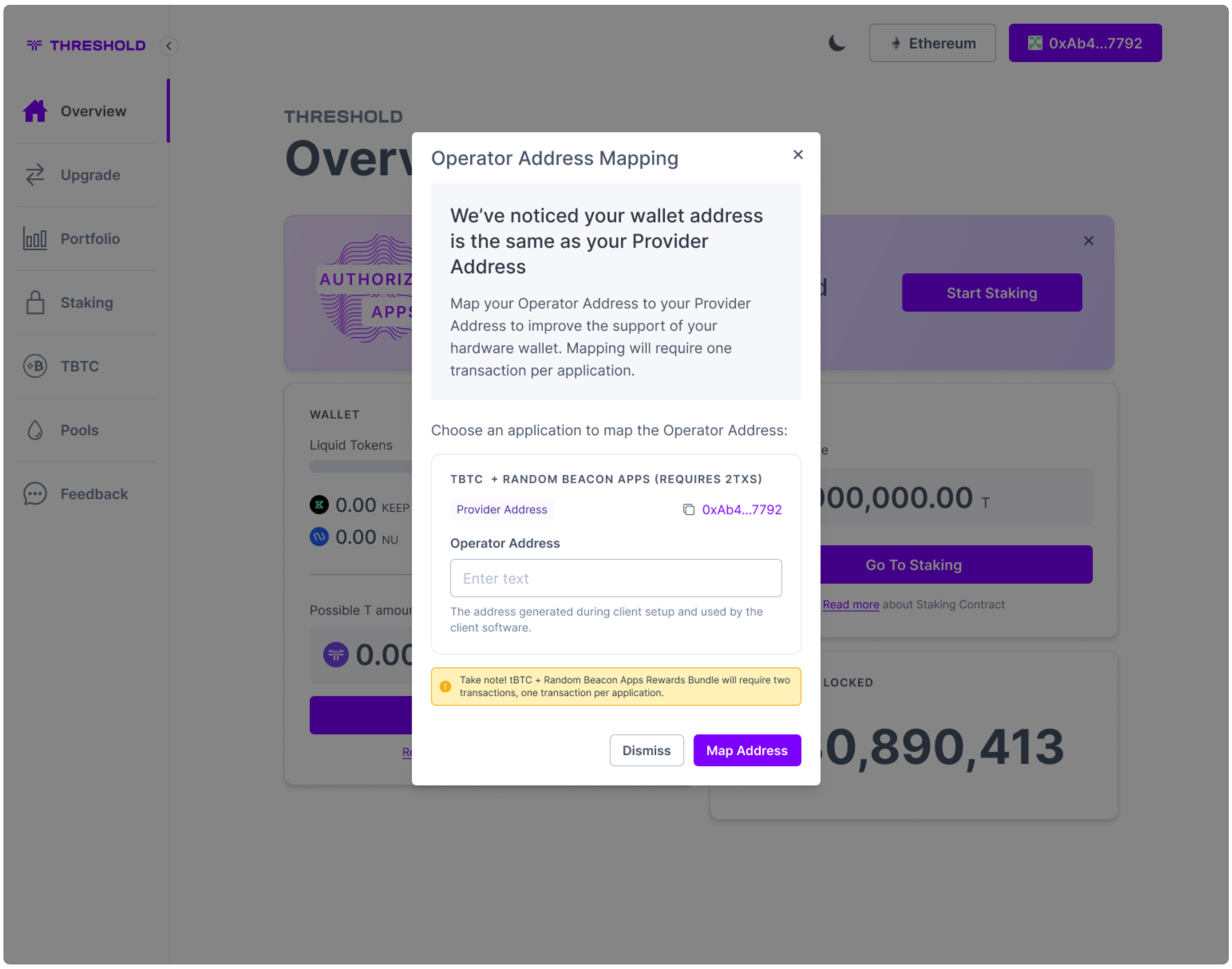
Minimum stake 40,000 T — Max

New Stake

Read More about T Staking

Registering Operator Address

After both applications have been authorized, click on "Start Mapping" to begin the Operator Registration process.


Operator Address Mapping

We've noticed your wallet address is the same as your Provider Address

Map your Operator Address to your Provider Address to improve the support of your hardware wallet. Mapping will require one transaction per application.

Choose an application to map the Operator Address:

TBTC + RANDOM BEACON APPS (REQUIRES 2TXS)

Provider Address: 0xAb4...7792

Operator Address: Enter text

The address generated during client setup and used by the client software.

Take note! tBTC + Random Beacon Apps Rewards Bundle will require two transactions, one transaction per application.

Registering Operator Address

Enter your Operator Address in the field provided and click on "Map Address."

Once the steps above have been successfully completed, you are ready to move on to the next step in the node deployment process.

Don't forget: a tBTC v2 node will not be able to be deployed without successfully authorizing both the tBTC and Random Beacon applications, and registering the node's operator address **FIRST**.

Network Configuration

Required network configuration for the tBTC v2 staking client.

Required Ports

The node has to be accessible publicly to establish and maintain connections with bootstrap nodes and discovered peers. The node exposes metrics and diagnostics services for network health monitoring purposes. Update firewall rules as necessary, including application level firewalls for the following ports

Purpose	Config Property	Protocol	Default
Network	network.port	TCP	3919
Status	clientInfo.port	TCP	9601

> The network port must be exposed publicly for peers to connect to your node.

> The status port must be exposed publicly for rewards allocation.

Announced Addresses

An Announced Address is a layered addressing information (`multiaddress`/`multiaddr`) announced to the Threshold Network that is used by peers to connect with your node, e.g.: `/dns4/bootstrap-0.test.keep.network/tcp/3919` or `/ip4/104.154.61.116/tcp/3919`.

If the machine you're running your node is not exposing a public IP (e.g. it is behind NAT) you should set the `network.AnnouncedAddresses` (flag: `--network.announcedAddresses`) configuration property to addresses (`ip4` or `dns4`) under which your node is reachable for the public.

To read more about `multiaddress` see the libp2p documentation.

Data Storage

Setup persistent data directories for the client.

The client requires two persistent directories. These directories will store configuration files and data generated and used by the client. It is highly recommended to create frequent backups of these directories. Loss of these data may be catastrophic and may lead to slashing.

> It is crucial to ensure the data directory is persisted and backed up on a regular basis.

Create folders for tBTC v2 client

```
cd /home
mkdir keep
cd keep
mkdir storage config
```

The tBTC v2 client will create two subdirectories within the storage directory: `keystore` and `work`. You do not need to create these.

The `keystore` subdirectory contains sensitive key material data generated by the client. Loosing the `keystore` data is a serious protocol offense and leads to slashing and potentially losing funds.

> It is the operator's responsibility to ensure the keystore data are not lost under any circumstances.

The `work` directory contains data generated by the client that should persist the client restarts or relocations. If the `work` data are lost the client will be able to recreate them, but it is inconvenient due to the time needed for the operation to complete and may lead to losing rewards.

Copy Operator keystore file

Assuming Geth was installed for the root user with the command provided in the Operator Account creation step, the operator-key file should be located in the `~/operator-key` directory.

```
cd ~/operator-key
```

Contained within the `operator-key` directory is the account key file (operator key file), its name will be similar to the following: `UTC--2018-11-01T06-23-57.810787758Z--fa3da235947aab49d439f3bcb46effd1a7237e32`

copy (not move!) this account key file to the `config` directory created above

```
ls -la
cp name_of_account_key_file /home/keep/config/name_of_account_key_file
```

Installation

A tBTC v2 node can be set up using either a docker or binary installation.

Docker Installation

This page will guide you through Docker setup steps.

Install Docker

Install or update Docker to the latest version. Visit the Official Docker website for detailed instructions. Use the command below to find your installed version if needed:

```
docker --version
```

Docker and Security Best Practices

General best practices recommend against running the tBTC v2 client as the `root` user as a security precaution. One security-minded approach is to Run the Docker daemon as a non-root user (Rootless mode).

Next, choose ONE of the following options. The **tBTC v2 Service** option configures the client to run as a service in order to ensure that the client is restarted automatically, should your machine reboot. The **Docker Launch Script** is faster to setup, but won't start the client if your machine reboots.

Systemd Service

Create the tbtcv2.service file:

```
cd /etc/systemd/system/
nano tbtcv2.service
```

Paste the following in the tbtcv2.service file:

```
[Unit]
Description=tBTC v2 client
After=network.target
Wants=network.target

[Service]
Environment="ETHEREUM_WS_URL=<Ethereum API WS URL>"
Environment="OPERATOR_KEY_FILE_NAME=<Operator Account keyfile name>"
Environment="OPERATOR_KEY_FILE_PASSWORD=<Operator Account keyfile password>"
Environment="PUBLIC_IP=/dns4/<PUBLIC_IP_OF_MACHINE>/tcp/3919"
Environment="CONFIG_DIR=/home/<user name>/keep/config"
Environment="STORAGE_DIR=/home/<user name>/keep/storage"

# These items only apply if you setup rootless-mode.
# Do not enable unless using rootless mode. Don't forget to adjust user UID.
#Environment="XDG_RUNTIME_DIR=/run/<user name>/<user UID>/"
#Environment="DOCKER_HOST=unix:///run/<user name>/<user UID>/docker.sock"

Type=simple
WorkingDirectory=/home/<user>

ExecStart=/usr/bin/docker run \
    --volume ${CONFIG_DIR}:/mnt/keep/config \
    --volume ${STORAGE_DIR}:/mnt/keep/storage \
    --env KEEP_ETHEREUM_PASSWORD=${OPERATOR_KEY_FILE_PASSWORD} \
    --env LOG_LEVEL=info \
    --log-opt max-size=100m \
    --log-opt max-file=3 \
    -p 3919:3919 \
    -p 9601:9601 \
    keepnetwork/keep-client:latest \
    start \
    --ethereum.url ${ETHEREUM_WS_URL} \
    --ethereum.keyFile /mnt/keep/config/${OPERATOR_KEY_FILE_NAME} \
    --storage.dir /mnt/keep/storage \
    --network.announcedAddresses $PUBLIC_IP

Restart=always
RestartSec=15s

[Install]
WantedBy=default.target
```

Replace the placeholders inside the <> brackets, remove the <> brackets but be sure to not edit anything further.

When done, save and close the file.

Next, test to make sure your configuration works:

```
sudo systemctl start tbtcv2
```

There will be no console output because it will be running in the background. Use systemctl to get the status of the service:

```
sudo systemctl status tbtcv2
```

If the service failed, go back and double check your configuration.

Another option is to see if the Docker container is running:

```
docker ps
```

If everything is running as intended, enable the service:

```
systemctl enable tbtcv2
```

Now, with the service running, you should make sure that your configuration will tolerate a reboot and start up again automatically.

```
reboot now
```

Log back in to your machine and check that the service is running. If it is, you're done. If not, go back and review your work.

Docker Container

Docker Launch Script

To launch the tBTC v2 client, several configuration flags and environmental values need to be set. For simplicity, a bash script can be used rather than typing or pasting all the flags into the console.

Create the launch script:

```
nano keep.sh
```

And paste the following:

```
# Keep tBTC v2 Client
#
# Ethereum endpoint WebSocket URL
# This can be a provider such as Infura, Alchemy, Ankr, etc or your own Geth Nodeq
# ETHEREUM_WS_URL="wss://mainnet.infura.io/ws/v3/redacted_credentials"
# note: only replace characters inside the " ". The Quotation marks must be retaine
ETHEREUM_WS_URL="<Ethereum API WS URL>"
```

```
# copied to home/keep/config earlier
OPERATOR_KEY_FILE_NAME="<Operator Account keyfile name>"

# password set during Operator Account Address creation
OPERATOR_KEY_FILE_PASSWORD="<Operator Account keyfile password>"

# To configure your node with a Public IP, enter it below.
PUBLIC_IP="<PUBLIC_IP_OF_MACHINE>"
# Alternatively, you can use DNS.
# To configure DNS, modify the last line of the script
# and add your DNS in the following format:
# /dns4/bootstrap-1.test.keep.network/tcp/3919

# Setup configuration and storage directories
# THESE MUST BE PERSISTENT STORAGE
CONFIG_DIR="/home/keep/config"
STORAGE_DIR="/home/keep/storage"

docker run \
    --detach \
    --restart on-failure \
    --volume $CONFIG_DIR:/mnt/keep/config \
    --volume $STORAGE_DIR:/mnt/keep/storage \
    --env KEEP_ETHEREUM_PASSWORD=$OPERATOR_KEY_FILE_PASSWORD \
    --env LOG_LEVEL=info \
    --log-opt max-size=100m \
    --log-opt max-file=3 \
    -p 3919:3919 \
    -p 9601:9601 \
    keepnetwork/keep-client:latest \
    start \
    --ethereum.url $ETHEREUM_WS_URL \
    --ethereum.keyFile /mnt/keep/config/$OPERATOR_KEY_FILE_NAME \
    --storage.dir /mnt/keep/storage \
    --network.announcedAddresses /ip4/$PUBLIC_IP/tcp/3919
```

Save and close the file, and make it executable:

```
sudo chmod +x keep.sh
```

To launch the tBTC v2 client, execute:

```
sudo bash keep.sh
```

ⓘ The `--detach` property will prevent the status messages from the client to be printed to the console. Review the Docker logs for detailed status information.

⚠ The path shown in the example configuration will differ from yours. Make sure it is configured correctly.

Client Startup

Unless the `--detach` flag was removed from the startup script, there will be no console output. In order to check your node, retrieve the Docker logs.

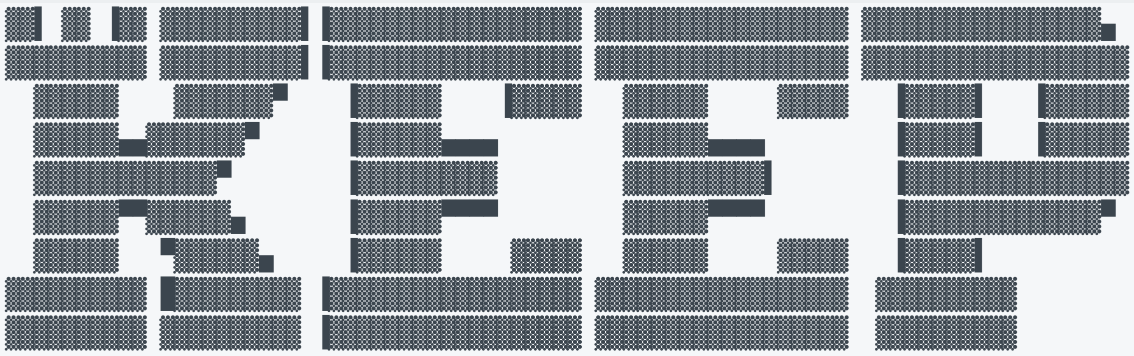
First, find your Docker instance identification, it'll be a random combination of words, e.g. `stinky_brownie`:

```
sudo docker ps
```

Use your specific identification and substitute:

```
sudo docker logs stinky_brownie >& /path/to/output/file
```

Scroll down about half a page, and you should see the following:



```
Trust math, not hardware.

-----------------------------------------------------------------------------------
| Keep Client Node                                                                |
|                                                                                 |
| Version: Version: v2.0.0-m1 (4d745f6d0)                                         |
|                                                                                 |
| Operator: 0x_your_operator_address                                              |
|                                                                                 |
| Port: 3919                                                                      |
| IPs : /ip4/111.222.333.444/tcp/3919/ipfs/redacted                               |
|                                                                                 |
| Contracts:                                                                      |
| RandomBeacon   : 0x5499f54b4A1CB4816eefCf78962040461be3D80b                     |
| WalletRegistry : 0x46d52E41C2F300BC82217Ce22b920c34995204eb                     |
| TokenStaking   : 0x01B67b1194C75264d06F808A921228a95C765dd7                     |
-----------------------------------------------------------------------------------
```

Congratulations, your node is up and running.

Binary Installation

This page will guide you through Binary setup steps.

> Choose either Docker installation OR Binary installation.

Download the Binary

Download the client binary file, make sure you are in the correct directory.

```
cd /home/keep
wget https://github.com/keep-network/keep-core/releases/download/v2.0.0-m1/keep-client-mainnet-v2.0.0-m1-linux-amd64.tar.gz
```

After the download completes, use the command below to display the contents of the folder:

```
ls -la
```

Extract the compressed file:

```
tar xzvf downloaded-file-name-here.tar.gz
```

> You can save some time and prevent misspelling a file name by typing the first few letter of the file name and pressing the Tab key. This will autocomplete the file name. Be sure to verify the file name prior to pressing the enter key.

Configure tBTC v2 Client

To launch the tBTC v2 client, several configuration flags and environmental values need to be set. For simplicity, a bash script can be used rather than typing or pasting all the flags into the console.

The following flags must be set at minimum:

```
--ethereum.url "wss://mainnet-ETH-enpoint-here"
--mainnet
--storage.dir "/home/keep/storage"
--ethereum.keyFile "/home/keep/config/UTC--Your-Operator-Key-Name"
# to configure your node to use your machine's public IP
--network.announcedAddresses "/ip4/your.ipv4.address.here/tcp/3919"
# to configure your node to use DNS, provide your DNS in the following
# format
# /dns4/bootstrap-1.test.keep.network/tcp/3919
```

For a complete list of client commands and flags, see CLI Options.

To launch the client, execute the following:

```
./keep-client start --ethereum.url "wss://mainnet-ETH-enpoint-here" --mainnet --storage.dir
"/home/keep/storage" --ethereum.keyFile "/home/keep/config/UTC--Your-Operator-Key-Name" --
network.announcedAddresses "/ip4/your.ipv4.address.here/tcp/3919"
```

If everything is configured correctly, the client will request the password for the Operator Account. Supply the password and press Enter.

You should see the following shortly:

```
▓▓▌ ▓▓ ▐▓▓ ▓▓▓▓▓▓▓▓▓▓▌▐▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓ ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓ ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▄
▓▓▓▓▓▓▓▓▓▓ ▓▓▓▓▓▓▓▓▓▓▌▐▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓ ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓ ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓
  ▓▓▓▓▓▓    ▓▓▓▓▓▓▓▀    ▐▓▓▓▓▓▓    ▐▓▓▓▓▓   ▓▓▓▓▓▓     ▓▓▓▓▓   ▐▓▓▓▓▓▌   ▐▓▓▓▓▓▓
  ▓▓▓▓▓▓▄▄▓▓▓▓▓▓▓▀      ▐▓▓▓▓▓▓▄▄▄▄         ▓▓▓▓▓▓▄▄▄▄         ▐▓▓▓▓▓▌   ▐▓▓▓▓▓▓
  ▓▓▓▓▓▓▓▓▓▓▓▓▓▀        ▐▓▓▓▓▓▓▓▓▓▓         ▓▓▓▓▓▓▓▓▓▓▌        ▐▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓
  ▓▓▓▓▓▓▀▀▓▓▓▓▓▓▄       ▐▓▓▓▓▓▓▀▀▀▀         ▓▓▓▓▓▓▀▀▀▀         ▐▓▓▓▓▓▓▓▓▓▓▓▓▓▓▀
  ▓▓▓▓▓▓   ▀▓▓▓▓▓▓▄     ▐▓▓▓▓▓▓     ▓▓▓▓▓   ▓▓▓▓▓▓     ▓▓▓▓▓   ▐▓▓▓▓▓▌
▓▓▓▓▓▓▓▓▓▓ █▓▓▓▓▓▓▓▓▓ ▐▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓ ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓  ▓▓▓▓▓▓▓▓▓▓
▓▓▓▓▓▓▓▓▓▓ ▓▓▓▓▓▓▓▓▓▓ ▐▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓ ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓  ▓▓▓▓▓▓▓▓▓▓

Trust math, not hardware.

-----------------------------------------------------------------------------------
| Keep Client Node                                                                |
|                                                                                 |
| Version: Version: v2.0.0-m1 (4d745f6d0)                                         |
|                                                                                 |
| Operator: 0x_your_operator_address                                              |
|                                                                                 |
| Port: 3919                                                                      |
| IPs : /ip4/111.222.333.444/tcp/3919/ipfs/redacted                               |
|                                                                                 |
| Contracts:                                                                      |
| RandomBeacon   : 0x5499f54b4A1CB4816eefCf78962040461be3D80b                     |
| WalletRegistry : 0x46d52E41C2F300BC82217Ce22b920c34995204eb                     |
| TokenStaking   : 0x01B67b1194C75264d06F808A921228a95C765dd7                     |
-----------------------------------------------------------------------------------
```

Congratulations, your node is up and running.

Advanced Options

Advanced configuration options and tBTC v2 staking client options

Logging

Configuration options for logging

Alternatives to Dashboard

Alternative application authorization methods to using the Threshold Dashboard.

Config File

Application configuration can be stored in a file and passed to the application with the `--config` flag.

CLI Options

Alternatives to Dashboard

Alternative application authorization methods to using the Threshold Dashboard.

An operator-registering transaction can be submitted with the Keep Client if the staking provider address key file is available.

Authorizing Random Beacon via the Client (Docker)

```
export KEEP_CLIENT_CONFIG_DIR=$(pwd)/config

export KEEP_CLIENT_ETHEREUM_WS_URL="<Ethereum API WS URL>"

export STAKING_PROVIDER_KEY_FILE_PASSWORD="<Staking Provider Account Key File Password>"
export STAKING_PROVIDER_KEY_FILE_NAME="<Staking Provider Account Key File Name>"
export OPERATOR_ADDRESS="<Operator Account Address>"

docker run \
    --volume $KEEP_CLIENT_CONFIG_DIR:/mnt/keep-client/config \
    --env KEEP_CLIENT_ETHEREUM_PASSWORD=$STAKING_PROVIDER_KEY_FILE_PASSWORD
    us-docker.pkg.dev/keep-test-f3e0/public/keep-client:latest \
    ethereum \
    --ethereum.url $KEEP_CLIENT_ETHEREUM_WS_URL \
    --ethereum.keyFile /mnt/keep-client/config/$STAKING_PROVIDER_KEY_FILE_NAME \
    beacon random-beacon register-operator --submit \
    $OPERATOR_ADDRESS
```

Authorizing TBTC application via the Client (Docker)

```
export KEEP_CLIENT_CONFIG_DIR=$(pwd)/config

export KEEP_CLIENT_ETHEREUM_WS_URL="<Ethereum API WS URL>"

export STAKING_PROVIDER_KEY_FILE_PASSWORD="<Staking Provider Account Key File Password>"
export STAKING_PROVIDER_KEY_FILE_NAME="<Staking Provider Account Key File Name>"
export OPERATOR_ADDRESS="<Operator Account Address>"

docker run \
    --volume $KEEP_CLIENT_CONFIG_DIR:/mnt/keep-client/config \
    --env KEEP_CLIENT_ETHEREUM_PASSWORD=$STAKING_PROVIDER_KEY_FILE_PASSWORD
    us-docker.pkg.dev/keep-test-f3e0/public/keep-client:latest \
    ethereum \

    --ethereum.url $KEEP_CLIENT_ETHEREUM_WS_URL \
    --ethereum.keyFile /mnt/keep-client/config/$STAKING_PROVIDER_KEY_FILE_NAME \
    ecdsa wallet-registry register-operator --submit
    $OPERATOR_ADDRESS
```

Via Web Browser

An operator-registering transactions can be submitted with Etherscan.

For each of the `RandomBeacon` and `WalletRegistry` contracts perform the following steps:

1. Find the address of the contract and open it on Etherscan (see below).
2. Go to `Contract` → `Write Contract` tab.
3. Connect your wallet with `Connect to Web3` button.
4. Submit the `registerOperator` function with your Operator address as an argument.

Logging

Optional logging configuration for the tBTC v2 staking client.

Configuration

Logging can be configured with environment variables. Please see sample settings:

```
LOG_LEVEL=DEBUG
IPFS_LOGGING_FMT=nocolor
GOLOG_FILE=/var/log/keep/keep.log
GOLOG_TRACING_FILE=/var/log/keep/trace.json
```

LOG_LEVEL option DEBUG will generate extensive output. Consider INFO instead.

If you want to share your LibP2P address with others you can get it from the startup log. When sharing remember to substitute the `/ipv4/` address with the public facing IP of your client if you're running on a private machine, or replace the entire `/ipv4/` segment with a DNS entry if you're using a hostname.

Config File

Use a config file to store client configuration.

Application configuration can be stored in a file and passed to the application with the `--config` flag.

Example:

```
./keep-client --config /path/to/your/config.toml start
```

Configuration files in formats TOML, YAML and JSON are supported.

Sample configuration file:

```toml
# This is a sample TOML configuration file for the Keep client.

[ethereum]
URL = "ws://127.0.0.1:8546"
KeyFile = "/Users/someuser/ethereum/data/keystore/UTC--2018-03-11T01-37-33.202765887Z--
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA8AAAAAAAAA"

# Uncomment to override the defaults for transaction status monitoring.

# MiningCheckInterval is the interval in which transaction
# mining status is checked. If the transaction is not mined within this
# time, the gas price is increased and transaction is resubmitted.
#
# MiningCheckInterval = 60  # 60 sec (default value)

# MaxGasFeeCap specifies the maximum gas fee cap the client is
# willing to pay for the transaction to be mined. The offered transaction
# gas cost can not be higher than the max gas fee cap value. If the maximum
# allowed gas fee cap is reached, no further resubmission attempts are
# performed. This property should be set only for Ethereum. In case of
# legacy non-EIP-1559 transactions, this field works in the same way as
# `MaxGasPrice` property.
#
# MaxGasFeeCap = "500 Gwei" # 500 Gwei (default value)

# Uncomment to enable Ethereum node rate limiting. Both properties can be
# used together or separately.
#
# RequestsPerSecondLimit sets the maximum average number of requests
# per second which can be executed against the Ethereum node.
# All types of Ethereum node requests are rate-limited,
# including view function calls.
#
# RequestsPerSecondLimit = 150

# ConcurrencyLimit sets the maximum number of concurrent requests which

# can be executed against the Ethereum node at the same time.
# This limit affects all types of Ethereum node requests,
# including view function calls.
#
# ConcurrencyLimit = 30

# BalanceAlertThreshold defines a minimum value of the operator's account
# balance below which the client will start reporting errors in logs.
# A value can be provided in `wei`, `Gwei` or `ether`, e.g. `7.5 ether`,
# `7500000000 Gwei`.
#
# BalanceAlertThreshold = "0.5 ether" # 0.5 ether (default value)

[network]
Bootstrap = false
```

```
Peers = [
"/ip4/127.0.0.1/tcp/3919/ipfs/16Uiu2HAmFRJtCWfdXhZEZHWb4tUpH1QMMgzH1oiamCfUuK6NgqWX
",
]
Port = 3920

# Uncomment to override the node's default addresses announced in the network
# AnnouncedAddresses = ["/dns4/example.com/tcp/3919", "/ip4/80.70.60.50/tcp/3919"]

# Uncomment to enable courtesy message dissemination for topics this node is
# not subscribed to. Messages will be forwarded to peers for the duration
# specified as a value in seconds.
# Message dissemination is disabled by default and should be enabled only
# on selected bootstrap nodes. It is not a good idea to enable dissemination
# on non-bootstrap node as it may clutter communication and eventually lead
# to blacklisting the node. The maximum allowed value is 90 seconds.
#
# DisseminationTime = 90

[storage]
Dir = "/my/secure/location"

# ClientInfo exposes metrics and diagnostics modules.
#
# Metrics collects and exposes information useful for external monitoring tools usually
# operating on time series data.
# All values exposed by metrics module are quantifiable or countable.
#
# The following metrics are available:
# - connected peers count
# - connected bootstraps count
# - eth client connectivity status
#
# Diagnostics module exposes the following information:
# - list of connected peers along with their network id and ethereum operator address
# - information about the client's network id and ethereum operator address
[clientInfo]
Port = 9601
# NetworkMetricsTick = 60
# EthereumMetricsTick = 600

# Uncomment to overwrite default values for TBTC config.
#
# [tbtc]
# PreParamsPoolSize = 3000
# PreParamsGenerationTimeout = "2m"
# PreParamsGenerationDelay = "10s"
# PreParamsGenerationConcurrency = 1
# KeyGenConcurrency = 1

# Developer options to work with locally deployed contracts
#
# [developer]
# TokenStakingAddress = "0xBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB"
# RandomBeaconAddress = "0xBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB"
# WalletRegistryAddress = "0xBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB"
# BridgeAddress = "0xBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB"
```

CLI Options

Review available CLI Options below.

keep-core/client-start-help at main · keep-network/keep-core
GitHub

```
$ keep-client start --help
Starts the Keep Client in the foreground

Usage:
  keep-client start [flags]

Flags:
      --ethereum.url string                        WS connection URL for Ethereum client.
      --ethereum.keyFile string                    The local filesystem path to Keep operator
      --ethereum.miningCheckInterval duration      The time interval in seconds in which trans
      --ethereum.maxGasFeeCap wei                  The maximum gas fee the client is willing t
      --ethereum.requestPerSecondLimit int         Request per second limit for all types of E
      --ethereum.concurrencyLimit int              The maximum number of concurrent requests w
      --ethereum.balanceAlertThreshold wei         The minimum balance of operator account bel
      --network.bootstrap                          Run the client in bootstrap mode.
      --network.peers strings                      Addresses of the network bootstrap nodes.
  -p, --network.port int                           Keep client listening port. (default 3919)
      --network.announcedAddresses strings         Overwrites the default Keep client address
      --network.disseminationTime int              Specifies courtesy message dissemination ti
      --storage.dir string                         Location to store the Keep client key share
      --clientInfo.port int                        Client Info HTTP server listening port. (de
      --clientInfo.networkMetricsTick duration     Client Info network metrics check tick in s
      --clientInfo.ethereumMetricsTick duration    Client info Ethereum metrics check tick in
      --tbtc.preParamsPoolSize int                 tECDSA pre-parameters pool size. (default 1
      --tbtc.preParamsGenerationTimeout duration   tECDSA pre-parameters generation timeout.
      --tbtc.preParamsGenerationDelay duration     tECDSA pre-parameters generation delay. (de
      --tbtc.preParamsGenerationConcurrency int    tECDSA pre-parameters generation concurrenc
      --tbtc.keyGenerationConcurrency int          tECDSA key generation concurrency. (default
      --developer.bridgeAddress string             Address of the Bridge smart contract
      --developer.randomBeaconAddress string       Address of the RandomBeacon smart contract
      --developer.tokenStakingAddress string       Address of the TokenStaking smart contract
      --developer.walletRegistryAddress string     Address of the WalletRegistry smart contrac

Global Flags:
  -c, --config string   Path to the configuration file. Supported formats: TOML, YAML, JSON.
      --developer       Developer network
      --goerli          Görli network
      --mainnet         Mainnet network

Environment variables:
    KEEP_ETHEREUM_PASSWORD    Password for Keep operator account keyfile decryption.
    LOG_LEVEL                 Space-delimited set of log level directives; set to "help" for h
```

Frequently Asked Questions

Find answers to some of the most commonly asked questions here.

Errors in Logs or Console

Certain errors may be reported by your node during the early stages of Chaosnet and the tBTC 2 launch. See a sample below:

```
2022-10-10T18:30:42.571Z    WARN    keep-beacon    beacon/node.go:78    selecting group not possible: [cannot select group in the sortition pool: [got error [execution reverted: Sortition pool unlocked] while resolving original error [execution reverted: Sortition pool unlocked]]]    {"seed": "0x29c60250c292e108e6abf4dcc76cb161d8ae8e803c4d4419eaff6e97f514b080"}
```

> This warning is normal and may happen on chaosnet. It will disappear and reappear periodically.

This is expected during this stage of the chaosnet. When the pool is locked, new operators cannot join but sortition (selecting operators from the pool) is possible. When the pool is unlocked, new operators can join but the sortition is not possible. Once the first set of **beta operators** is registered, the pool will get locked for some time to allow the sortition. After some time, when another set of beta operators are added, the pool will be unlocked again.

Staking Providers

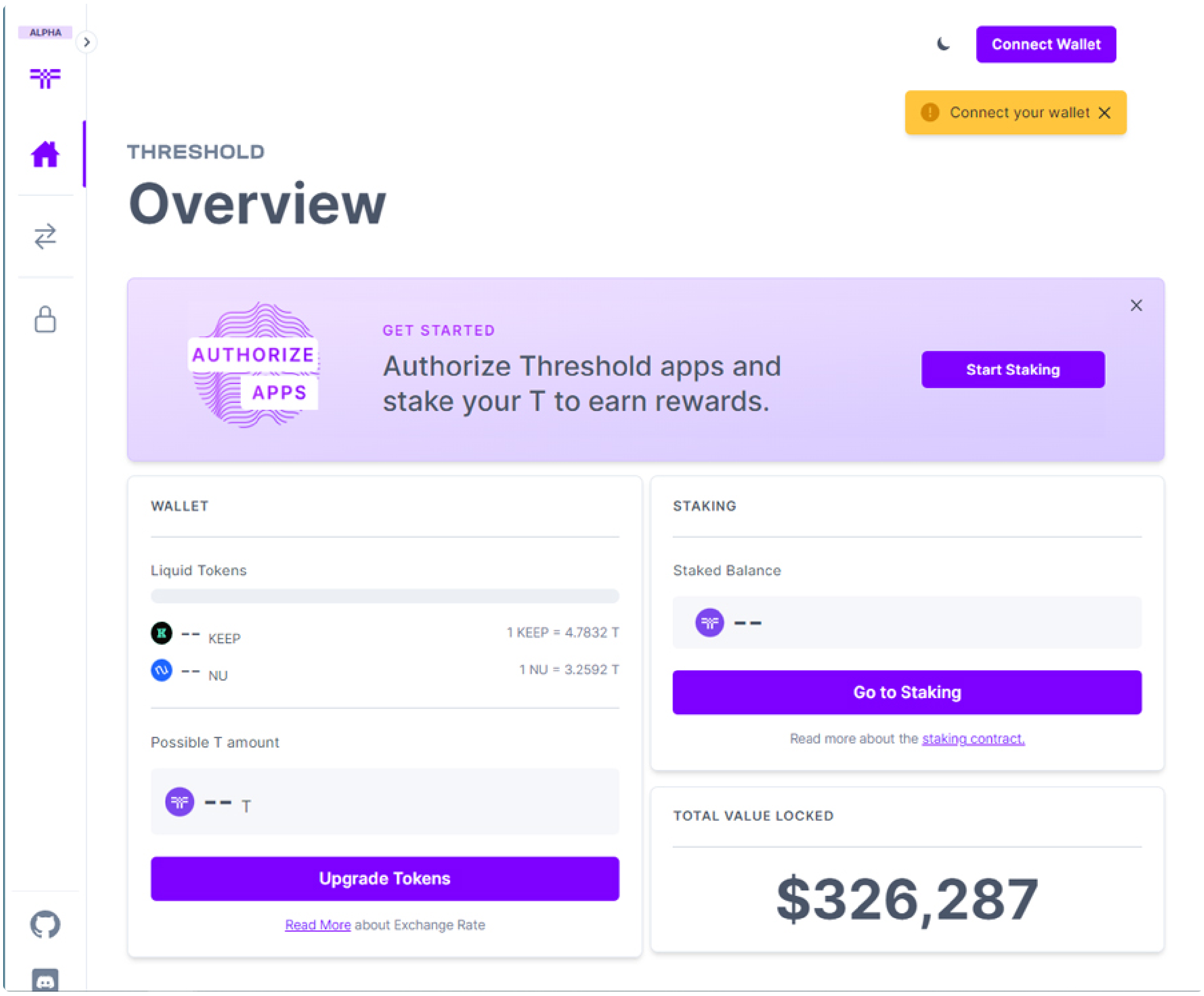
You can delegate running an application node to one of the *node-as-a-service* Staking Providers listed below.

Staking Providers have not been vetted or endorsed by Threshold. Use your judgement when selecting a provider.

Staking Provider	Contact Information
Ankr	sales@ankr.com
Coinbase Cloud	cloud-sales@coinbase.com
Blockdaemon	konstantin@blockdaemon.com
Boar	hello@boar.network
DELIGHT	contact@delightlabs.io
Figment	support@figment.io
InfStones	sales@infstones.com
Low Fee Validation	eduardo@lowfeevalidation.com
P2P	am@p2p.org
Staked	staked@staked.us

Goerli Testnet

Here you'll find the Goerli testnet version of the Threshold dapp.


ALPHA

Connect Wallet

Connect your wallet

THRESHOLD

Overview

GET STARTED

Authorize Threshold apps and stake your T to earn rewards.

Start Staking

WALLET

Liquid Tokens

-- KEEP 1 KEEP = 4.7832 T

-- NU 1 NU = 3.2592 T

Possible T amount

-- T

Upgrade Tokens

Read More about Exchange Rate

STAKING

Staked Balance

--

Go to Staking

Read more about the staking contract.

TOTAL VALUE LOCKED

$326,287

Goerli Testnet - Threshold Dashboard

Testnet tBTC v2 node Setup

This page will show you how to launch a tBTC v2 node on the testnet.

> ⚠ This is a TESTNET guide document. Following this document will not result in a node enabling you to earn mainnet rewards.

Recommended Machine Types

> ⓘ While it is possible to run the client on a local machine, this is not recommended.

Your operating environment will ultimately dictate what machine type to go with. This is particularly relevant if you're running a containerized solution where multiple applications are sharing VM resources. The below types are sufficient for running one instance of the tBTC v2 Node.

The preferred OS is Ubuntu.

VPS Provider	VPS Type
AWS	c5.large
Azure	F2s v2
Google Cloud	n2-highcpu-2
Self-hosted	2 vCPU / 2 GB RAM / 1 GiB Persistent Storage

Ethereum API

A Keep Node requires a connection to a WebSocket Ethereum API. You should obtain a WS API URL from a service provider (e.g. Alchemy, Infura, Ankr) or run your own Ethereum node (e.g. Geth).

The client requires an Ethereum Key File of an Operator Account to connect to the Ethereum chain. This account is created in a subsequent step using Geth (GoEthereum).

The Ethereum Key File is expected to be encrypted with a password. The password has to be provided in a prompt after the client starts or configured as a `KEEP_ETHEREUM_PASSWORD` environment variable.

The Operator Account has to maintain a positive Ether balance at all times.

> ⓘ Please do NOT reuse an operator account that is being used for PRE or other applications.

Install Geth (GoEthereum)

To create a new Ethereum account, install Geth (GoEthereum) and create a new account using the command below. This account will subsequently be referred to as the Operator Account.

```
geth account new --keystore ./operator-key
```

When prompted, provide a password to protect the operator key file.

> Use a password manager to generate a strong password and store it safely. It will be needed again during setup.

> Avoid passwords that contain the following characters: ', ", `, $
> These characters may be interpreted as part of the configuration which can lead to undesirable outcomes that may be extremely time intensive to correct.

Once the process completes, your public key will be displayed. Take note of your Operator Account public key.

> DO NOT LOSE THE PASSWORD TO THE OPERATOR ACCOUNT.

Funding your Operator Account

Your Operator Account will need to be funded with goerli ETH and maintain a positive balance at all times to ensure proper operation and availability of your tBTC v2 node.

Network Configuration

The node has to be accessible publicly to establish and maintain connections with bootstrap nodes and discovered peers.

> Update firewall rules as necessary, including application level firewalls.

The node exposes metrics and diagnostics services for monitoring. A network port has to be exposed publicly, so the peers can connect to your node. A Diagnostics Port has to be exposed publicly, for the rewards allocation.

Purpose	Config Property	Protocol	Default
Network	network.port	TCP	3919
Status	clientInfo.port	TCP	9601

A Diagnostics Port has to be exposed publicly, for the rewards allocation.

Announced Addresses

An Announced Address is a layered addressing information (`multiaddress`/`multiaddr`) announced to the Threshold Network that is used by peers to connect with your node, e.g.: `/dns4/bootstrap-0.test.keep.network/tcp/3919` or `/ip4/104.154.61.116/tcp/3919`.

If the machine you're running your node is not exposing a public IP (e.g. it is behind NAT) you should set the `network.AnnouncedAddresses` (flag: `--network.announcedAddresses`) configuration property to an addresses (`ip4` or `dns4`) under which your node is reachable for the public.

To read more about `multiaddress` see the libp2p docummentation.

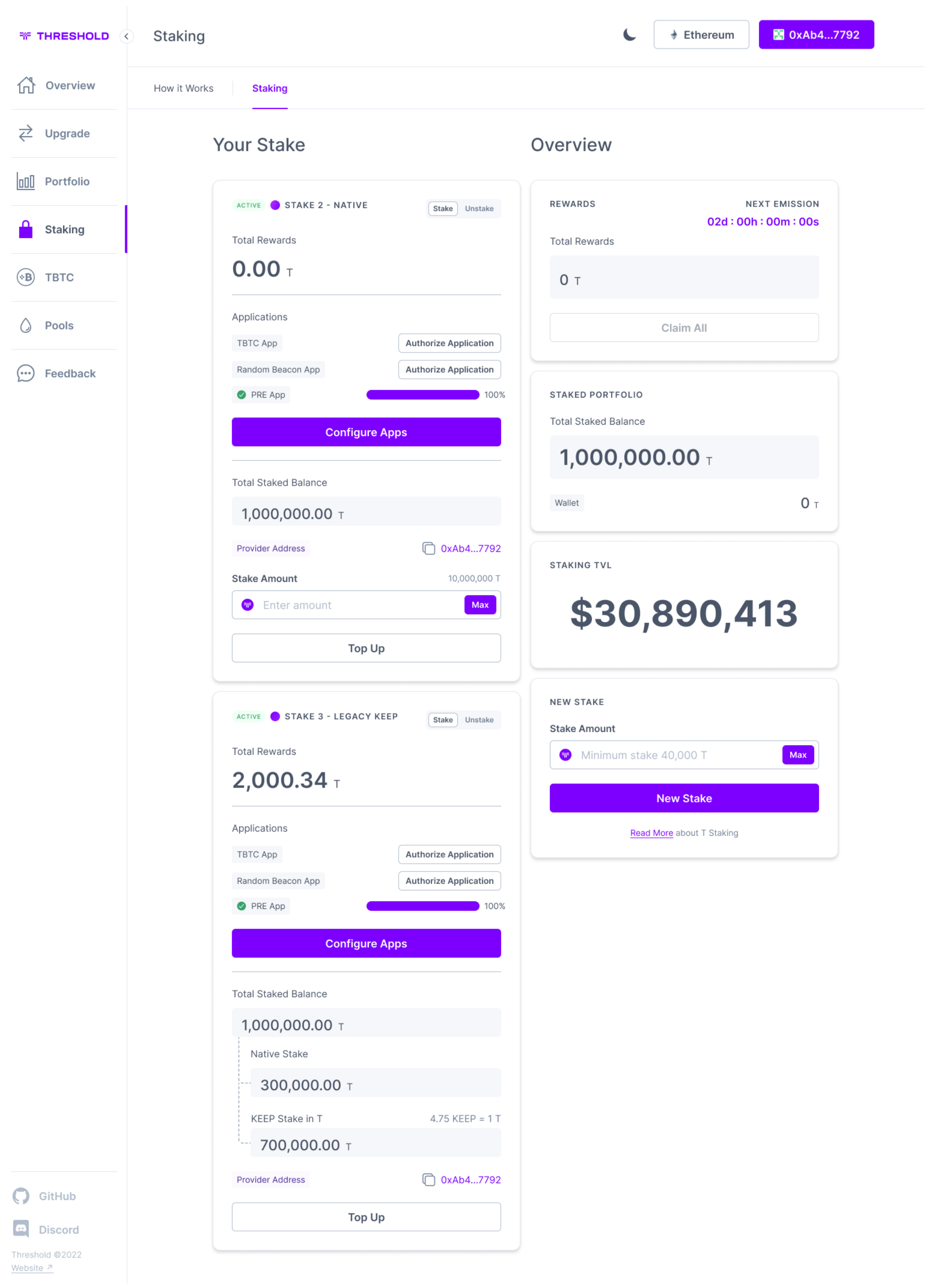
One important step to get your node operating on the Threshold Network is proper application authorization as well as operator account registration. Applications need only be authorized once.

It is **CRITICALLY** important that both the tBTC application as well as the Random Beacon applications are authorized. A node cannot be deployed without both applications being properly authorized.

Please note: by authorizing these applications, an unbonding period of 45 days will go into effect on the T you stake for these applications. This cool-down period begins the day you submit an unstake request.

Application Authorization

To get started, visit the Threshold Dashboard and connect your wallet.



THRESHOLD

Staging

Ethereum

0xAb4...7792

Overview

Upgrade

Portfolio

Staking

TBTC

Pools

Feedback

How it Works

Staking

Your Stake

ACTIVE STAKE 2 - NATIVE

Stake Unstake

Total Rewards

0.00 T

Applications

TBTC App	Authorize Application
Random Beacon App	Authorize Application
PRE App	100%

Configure Apps

Total Staked Balance

1,000,000.00 T

Provider Address 0xAb4...7792

Stake Amount 10,000,000 T

Enter amount Max

Top Up

ACTIVE STAKE 3 - LEGACY KEEP

Stake Unstake

Total Rewards

2,000.34 T

Applications

TBTC App	Authorize Application
Random Beacon App	Authorize Application
PRE App	100%

Configure Apps

Total Staked Balance

1,000,000.00 T

Native Stake

300,000.00 T

KEEP Stake in T 4.75 KEEP = 1 T

700,000.00 T

Provider Address 0xAb4...7792

Top Up

Overview

REWARDS

NEXT EMISSION

02d : 00h : 00m : 00s

Total Rewards

0 T

Claim All

STAKED PORTFOLIO

Total Staked Balance

1,000,000.00 T

Wallet 0 T

STAKING TVL

$30,890,413

NEW STAKE

Stake Amount

Minimum stake 40,000 T Max

New Stake

Read More about T Staking

GitHub

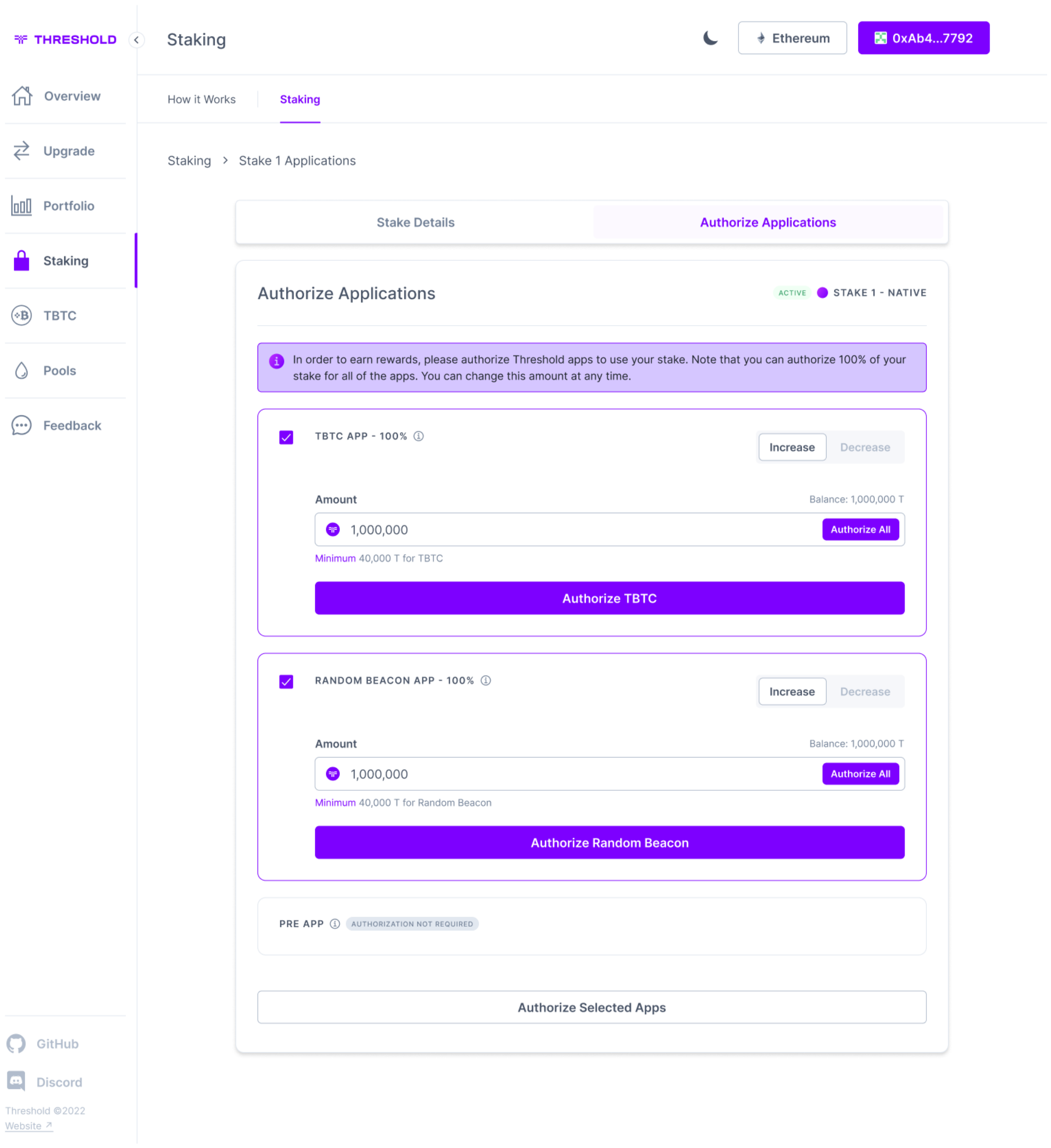
Discord

Threshold ©2022

Website ↗

Threshold Dashboard

Click on "Configure Apps"


THRESHOLD

Staking

Ethereum | 0xAb4...7792

Overview
Upgrade
Portfolio
Staking
TBTC
Pools
Feedback

How it Works | Staking

Staking > Stake 1 Applications

Stake Details | Authorize Applications

Authorize Applications

ACTIVE ● STAKE 1 - NATIVE

In order to earn rewards, please authorize Threshold apps to use your stake. Note that you can authorize 100% of your stake for all of the apps. You can change this amount at any time.

✓ TBTC APP - 100%

Increase | Decrease

Amount — Balance: 1,000,000 T

1,000,000 — Authorize All

Minimum 40,000 T for TBTC

Authorize TBTC

✓ RANDOM BEACON APP - 100%

Increase | Decrease

Amount — Balance: 1,000,000 T

1,000,000 — Authorize All

Minimum 40,000 T for Random Beacon

Authorize Random Beacon

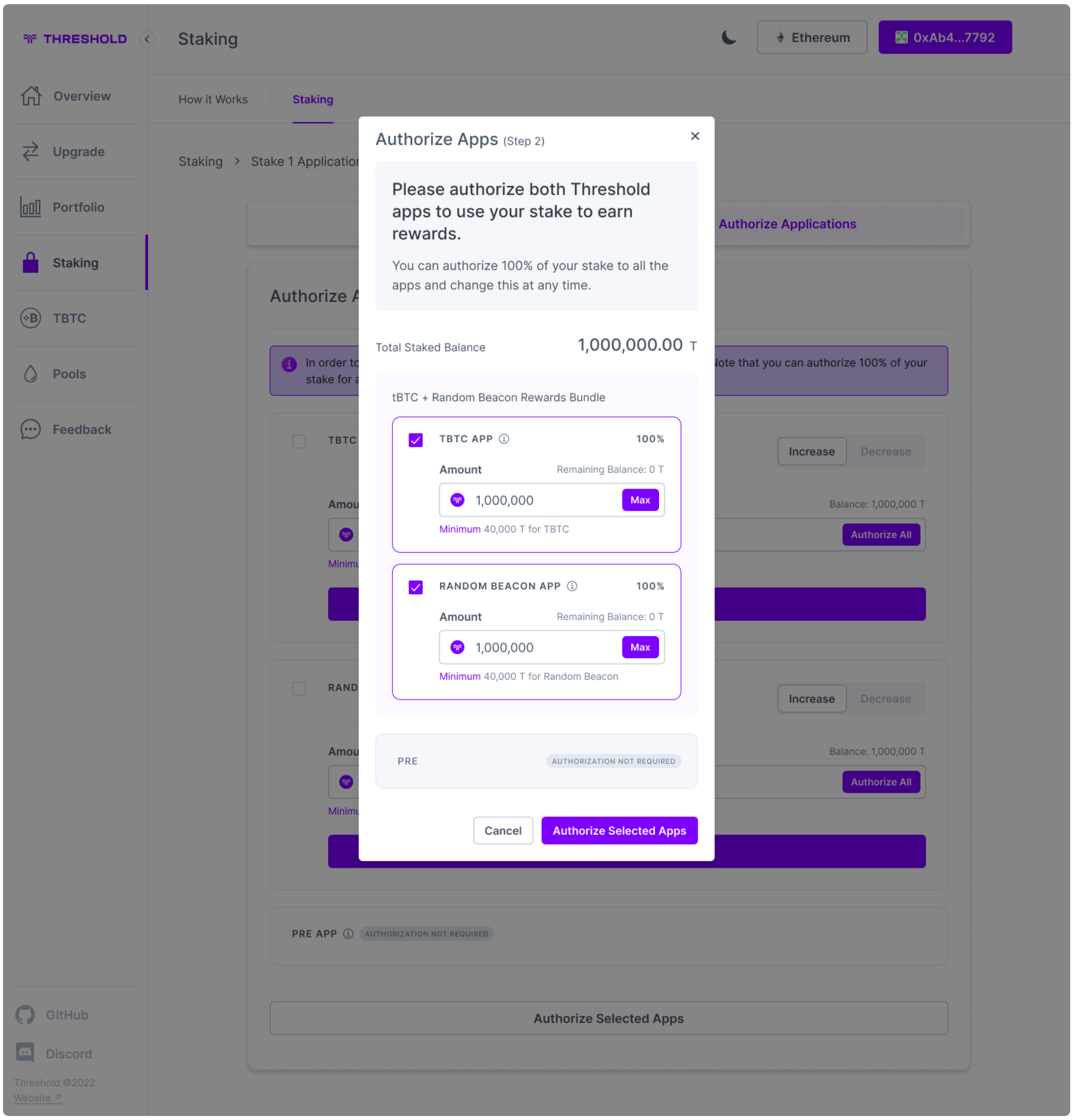
PRE APP — AUTHORIZATION NOT REQUIRED

Authorize Selected Apps

GitHub
Discord
Threshold ©2022
Website ↗

Authorizing Threshold Applications

Select BOTH tBTC and Random Beacon applications and enter your desired amount of T to stake per application. Note that the minimum is 40,000T.



Authorize Apps (Step 2)

Please authorize both Threshold apps to use your stake to earn rewards.

You can authorize 100% of your stake to all the apps and change this at any time.

Total Staked Balance 1,000,000.00 T

tBTC + Random Beacon Rewards Bundle

☑ TBTC APP 100%
Amount — Remaining Balance: 0 T
1,000,000 Max
Minimum 40,000 T for TBTC

☑ RANDOM BEACON APP 100%
Amount — Remaining Balance: 0 T
1,000,000 Max
Minimum 40,000 T for Random Beacon

PRE — AUTHORIZATION NOT REQUIRED

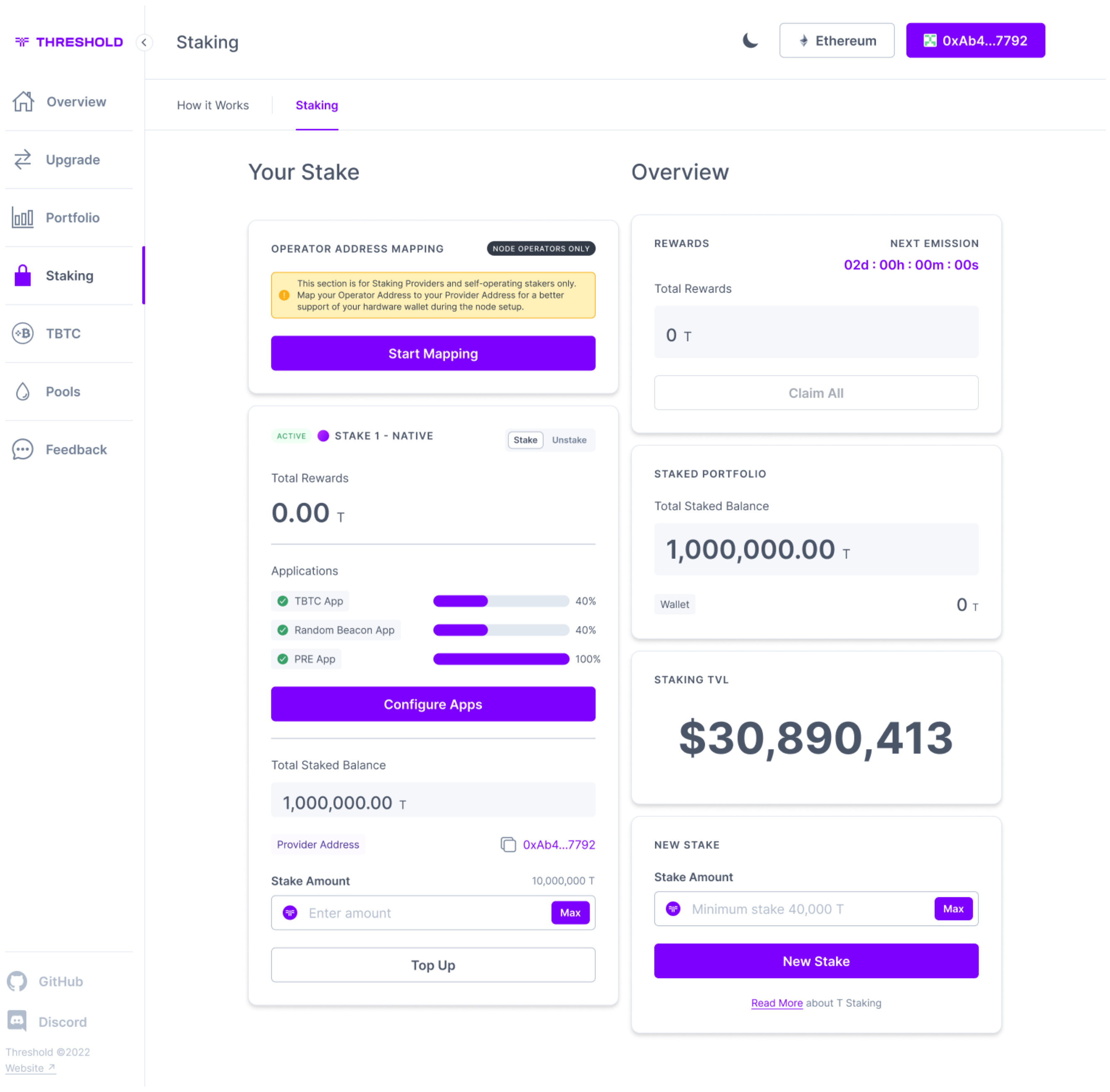
Cancel | Authorize Selected Apps

Authorizing Threshold Applications

Operator Registration

The operator account is the Ethereum account created on your node. In order for the network to associate your T stake with your node, you must register your Operator Address.

> ⓘ An Operator for the Provider registration can be submitted just once. The Operator address assignment cannot be updated.



Your Stake

OPERATOR ADDRESS MAPPING — NODE OPERATORS ONLY

This section is for Staking Providers and self-operating stakers only. Map your Operator Address to your Provider Address for a better support of your hardware wallet during the node setup.

Start Mapping

ACTIVE ● STAKE 1 - NATIVE — Stake | Unstake

Total Rewards
0.00 T

Applications

✓ TBTC App	40%
✓ Random Beacon App	40%
✓ PRE App	100%

Configure Apps

Total Staked Balance
1,000,000.00 T

Provider Address — 0xAb4...7792

Stake Amount — 10,000,000 T
Enter amount — Max

Top Up

Overview

REWARDS — NEXT EMISSION 02d : 00h : 00m : 00s

Total Rewards
0 T

Claim All

STAKED PORTFOLIO

Total Staked Balance
1,000,000.00 T

Wallet — 0 T

STAKING TVL
$30,890,413

NEW STAKE

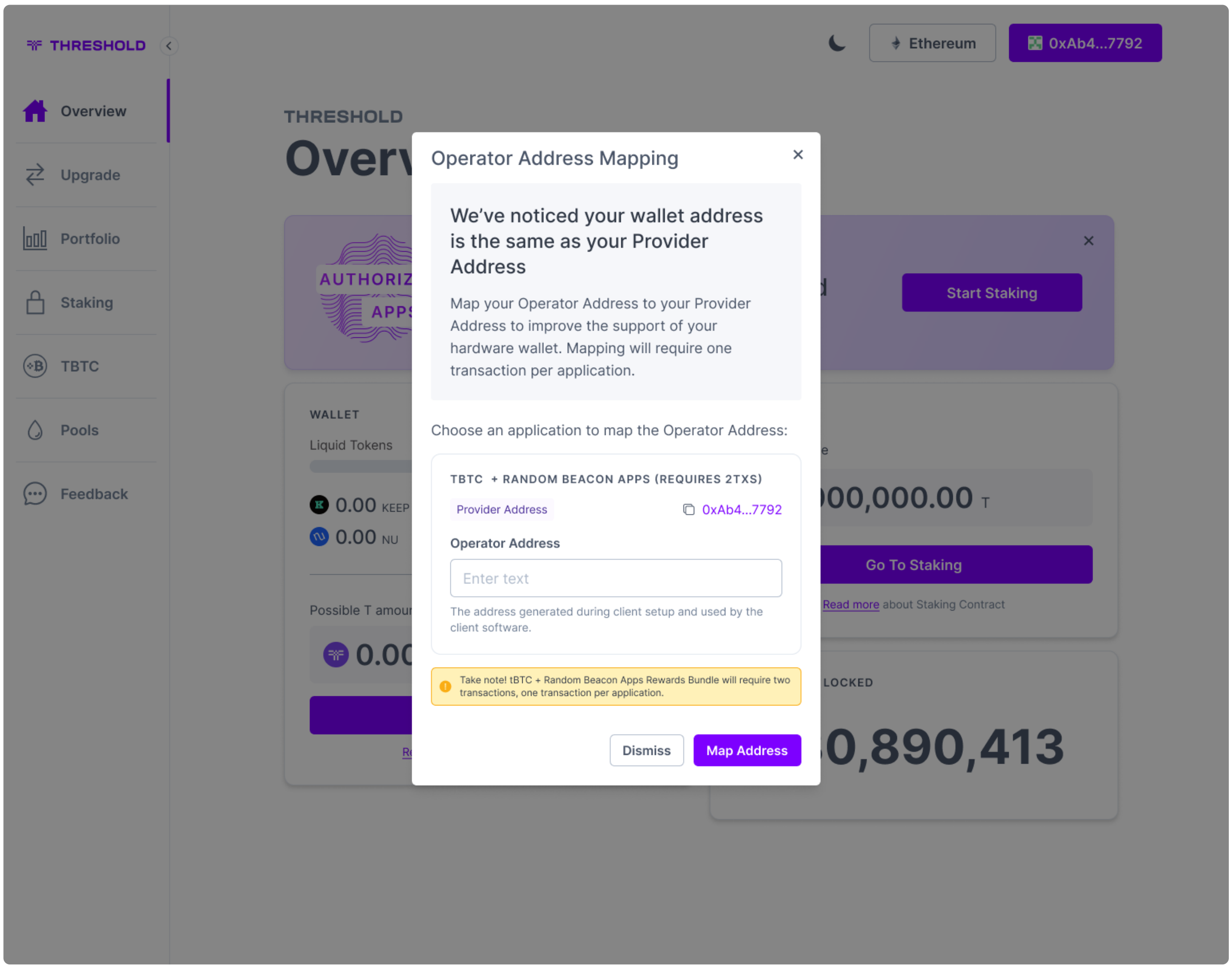
Stake Amount
Minimum stake 40,000 T — Max

New Stake

Read More about T Staking

Registering Operator Address

After both applications have been authorized, click on "Start Mapping" to begin the Operator Registration process.


Operator Address Mapping

We've noticed your wallet address is the same as your Provider Address

Map your Operator Address to your Provider Address to improve the support of your hardware wallet. Mapping will require one transaction per application.

Choose an application to map the Operator Address:

TBTC + RANDOM BEACON APPS (REQUIRES 2TXS)

Provider Address 0xAb4...7792

Operator Address

Enter text

The address generated during client setup and used by the client software.

Take note! tBTC + Random Beacon Apps Rewards Bundle will require two transactions, one transaction per application.

Registering Operator Address

Enter your Operator Address in the field provided and click on "Map Address."

Once the steps above have been successfully completed, you are ready to move on to the next step in the node deployment process.

Don't forget: a tBTC v2 node will not be able to be deployed without successfully authorizing both the tBTC and Random Beacon applications, and registering the node's operator address **FIRST**.

Create Folder Structure

The client requires two persistent directories. These directories will store configuration files and data generated and used by the client. It is highly recommended to create frequent backups of these directories. Loss of these data may be catastrophic and may lead to slashing.

It is crucial to ensure the data directory is persisted and backed up on a regular basis.

Create folders for tBTC v2 client

```
cd /home
mkdir keep
cd keep
mkdir storage config
```

The tBTC v2 client will create two subdirectories within the storage directory: `keystore` and `work`. You do not need to create these.

The `keystore` subdirectory contains sensitive key material data generated by the client. Loosing the `keystore` data is a serious protocol offense and leads to slashing and potentially losing funds.

> ⓘ It is the operator's responsibility to ensure the keystore data are not lost under any circumstances.

The `work` directory contains data generated by the client that should persist the client restarts or relocations. If the `work` data are lost the client will be able to recreate them, but it is inconvenient due to the time needed for the operation to complete and may lead to losing rewards.

Copy Operator keystore file

Assuming Geth was installed for the root user with the command provided in the Operator Account creation step, the operator-key file should be located in the `~/operator-key` directory.

```
cd ~/operator-key
```

Contained within the `operator-key` directory is the account key file (operator key file), its name will be similar to the following: `UTC--2018-11-01T06-23-57.810787758Z--fa3da235947aab49d439f3bcb46effd1a7237e32`

copy (not move!) this account key file to the `config` directory created above

```
ls -la
cp name_of_account_key_file /home/keep/config/name_of_account_key_file
```

Install Docker

Install or update Docker to the latest version. Visit the Official Docker website for detailed instructions. Use the command below to find your installed version if needed:

```
docker --version
```

Docker Launch Script

To launch the tBTC v2 client, several configuration flags and environmental values need to be set. For simplicity, a bash script can be used rather than typing or pasting all the flags into the console.

Create the launch script:

```
nano keep.sh
```

And paste the following:

```
# Keep Testnet tBTC v2 Client
#
# Ethereum endpoint WebSocket URL
# This can be a provider such as Infura, Alchemy, Ankr, etc or your own Geth Nodeq
# ETHEREUM_WS_URL="wss://goerli.infura.io/ws/v3/redacted_credentials"
# note: only replace characters inside the " ". The Quotation marks must be retained
ETHEREUM_WS_URL="<Ethereum API WS URL>"

# copied to home/keep/config earlier
OPERATOR_KEY_FILE_NAME="<Operator Account keyfile name>"

# password set during Operator Account Address creation
OPERATOR_KEY_FILE_PASSWORD="<Operator Account keyfile password>"

# To configure your node with a Public IP, enter it below.
PUBLIC_IP="<PUBLIC_IP_OF_MACHINE>"
# Alternatively, you can use DNS.
# To configure DNS, modify the last line of the script
# and add your DNS in the following format:
# /dns4/bootstrap-1.test.keep.network/tcp/3919

# Setup configuration and storage directories
# THESE MUST BE PERSISTENT STORAGE
CONFIG_DIR="/home/keep/config"
STORAGE_DIR="/home/keep/storage"

docker run \
    --detach \
    --restart on-failure \
    --volume $CONFIG_DIR:/mnt/keep/config \
    --volume $STORAGE_DIR:/mnt/keep/storage \
    --env KEEP_ETHEREUM_PASSWORD=$OPERATOR_KEY_FILE_PASSWORD \
    --env LOG_LEVEL=info \
    --log-opt max-size=100m \
    --log-opt max-file=3 \
    -p 3919:3919 \
    -p 9601:9601 \
    us-docker.pkg.dev/keep-test-f3e0/public/keep-client \
    start \
    --goerli \
    --ethereum.url $ETHEREUM_WS_URL \
    --ethereum.keyFile /mnt/keep/config/$OPERATOR_KEY_FILE_NAME \
    --storage.dir /mnt/keep/storage \
    --network.announcedAddresses /ip4/$PUBLIC_IP/tcp/3919
```

Save and close the file, and make it executable:

```
sudo chmod +x keep.sh
```

To launch the tBTC v2 client, execute:

```
sudo bash keep.sh
```

ⓘ The `--detach` property will prevent the status messages from the client to be printed to the console. Review the Docker logs for detailed status information.

⚠ The path shown in the example configuration will differ from yours. Make sure it is configured correctly.

Client Startup

Unless the `--detach` flag was removed from the startup script, there will be no console output. In order to check your node, retrieve the Docker logs.

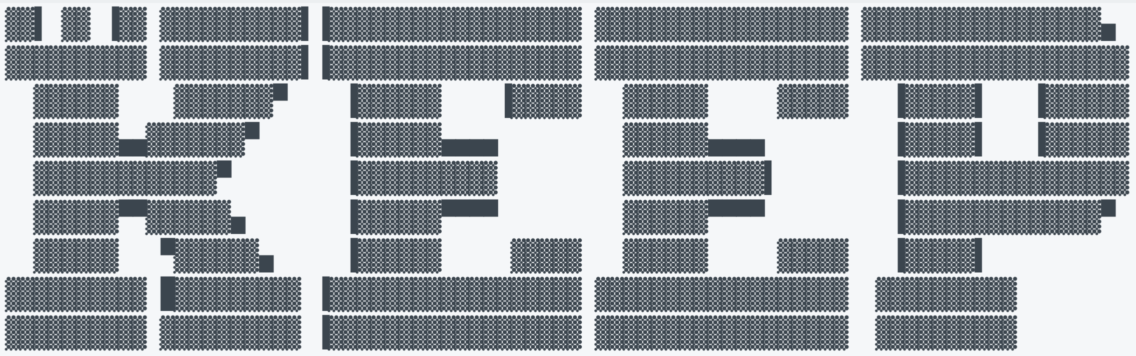
First, find your Docker instance identification, it'll be a random combination of words, e.g. `stinky_brownie`:

```
sudo docker ps
```

Use your specific identification and substitute:

```
sudo docker logs stinky_brownie >& /path/to/output/file
```

Scroll down about half a page, and you should see the following:



```
Trust math, not hardware.

-----------------------------------------------------------------------------------
| Keep Client Node                                                                |
|                                                                                 |
| Version: Version: v2.0.0-m1 (4d745f6d0)                                         |
|                                                                                 |

| Operator: 0x_your_operator_address                                              |
|                                                                                 |
| Port: 3919                                                                      |
| IPs : /ip4/111.222.333.444/tcp/3919/ipfs/redacted                               |
|                                                                                 |
| Contracts:                                                                      |
| RandomBeacon   : 0x2bA82903B635a96154A515488d2952E86D6adc3A                     |
| WalletRegistry : 0x2363cc10b7680000C02E4a7067A68d1788ffc86F                     |
| TokenStaking   : 0x69f962a0fbA5635e84eC94131f9072108E2E4F24                     |
-----------------------------------------------------------------------------------
```

Congratulations, your node is up and running.

App Development

Threshold Access Control

Use Threshold Access Control for end-to-end encrypted, end-to-end decentralized data sharing and communication.

-> See **Threshold Access Control** in the Fundamentals section for an overview of the over-arching concepts and value propositions.

There are multiple products and versions under the umbrella of 'Threshold Access Control':

- Conditions-Based Decryption, **Proof-of-Concept**
  Use this version to familiarize yourself with the SDK, API and developer-facing configuration/parametrization. Get started here.
- Conditions-Based Decryption, **Mainnet version**
  This version is under intense development and will be released in early Q2 2022. Learn about the vision for the product here and underlying trust model here.
- Proxy Re-encryption, **Mainnet version**
  This product and version is currently live on the Threshold Mainnet and has been battle-tested for over 2 years. Learn the fundamentals of the PRE offering here and get started building with PRE here.

Get Started (CBD PoC)

This tutorial is a quick way for developers to learn about the Threshold Access Control service by building with the **Proof-of-Concept (PoC)** version of Conditions-Based Decryption.

Note that the underlying trust assumptions vary between versions and technologies; these are explained in detail for the CBD Proof-of-Concept, CBD Mainnet, and PRE Mainnet versions.

1. Install `nucypher-ts`

`nucypher-ts` is under active development.

To begin, we need to install the `nucypher-ts` library:

```
yarn add @nucypher/nucypher-ts
```

One of the `nucypher-ts` dependencies takes advantage of WASM. In order to run `nucypher-ts` in the browser, we have to load WASM from the source files. This process is mostly automated by the wrapper generated by `wasm-pack`.

For this tutorial we'll need a few extra packages:

```
yarn add ethers @metamask/detect-provider
```

Visit `nucypher-ts/examples` for more information on using `nucypher-ts` in your web application.

2. Build a Cohort

Next, we will create a `Cohort` based on our risk preferences. A `Cohort` is a group of nodes that work together to control access to data. Threshold and Shares are two parameters used to create a `Cohort`. For example, a 3-of-5 `Cohort` needs at least 3 of the 5 members to provide shares to access the original data.

To create a `Cohort`, use the following code:

```
import { Cohort } from '@nucypher/nucypher-ts';

const config = {
  threshold: 3,
  shares: 5,
  porterUri: 'https://porter-tapir.nucypher.community',
};
const newCohort = await Cohort.create(config);
```

Notice that we provided a `porterUri` parameter. Porter is a web-based service that interacts with nodes on the network on behalf of applications. It acts as an "Infura for Threshold Access Control". In this example, we used a Porter endpoint for the `tapir` testnet.

3. Specify default Conditions

Conditions are the requirements for a data requester or recipient to access the plaintext data – i.e. what they will need to prove later to gain decryption rights.

The `ERC721Ownership` condition checks the owner of a given token ID. It can be customized by using the `ownerOf` contract method and comparing it with the requestor's signature. For more information, see the References section.

> ⓘ CBD allows developers to enforce conditional access at various runtime stages, depending on what makes the most sense for the use case. At this point, we will add the *default* Conditions, which will only gate-keep access *if no other conditions are included later at encryption time*. This is explained further in Condition Hierarchies.

We will now specify the conditions that must be met to access the data. In this tutorial, we will require that the requester owns an ERC721 token with a token ID of 5954.

```
import { Conditions } from '@nucypher/nucypher-ts';

const NFTOwnership = new Conditions.ERC721Ownership({
  contractAddress: '0xBC4CA0EdA7647A8aB7C2061c2E118A18a936f13D',
  chain: 5, // Tapir network uses Görli testnet
  parameters: [5954],
});
```

> ⓘ There are multiple Condition types and it is possible to combine multiple conditions into a ConditionSet.

```
import { Conditions, ConditionSet } from '@nucypher/nucypher-ts';

const conditions = new ConditionSet([
  NFTOwnership,
  // Other conditions can be added here
]);
```

4. Build a Strategy

We will now combine the `Cohort`, `ConditionSet`, and any other necessary parameters into a `Strategy`. Strategies are a convenient way to bundle together frequently used configurations, including specific combinations of network parameters and conditionality.

```
import { Strategy } from '@nucypher/nucypher-ts';

const newStrategy = Strategy.create(newCohort, conditions);
```

Next, we will deploy this `Strategy` to the Threshold Network. To do that, we're going to transact on Polygon Mumbai:

```
import detectEthereumProvider from '@metamask/detect-provider';
import { providers } from 'ethers';

const MMprovider = await detectEthereumProvider();
const mumbai = providers.getNetwork(80001);

const web3Provider = new providers.Web3Provider(MMprovider, mumbai);
const newDeployed = await newStrategy.deploy('test', web3Provider);
```

Deploying a `Strategy` requires writing to the blockchain. This requires a wallet funded with testnet MATIC and connection to the blockchain via a `provider` (e.g. MetaMask).

For more information about customizing and reusing `Cohort`, `Condition`, and `Strategy` objects, see the References page in the documentation.

5. Encrypt the plaintext & update Conditions

We can now encrypt data using the newly deployed `Strategy`. At this point, we can also specify new conditions on which data access will be predicated. These will take a higher precedence and override the default conditions contained in the Strategy. For this example, let's make it so the overriding requirement is that the requester's wallet hold a minimum number (three) of NFTs. The NFTs are from the same collection that we specified in Step 3. Note that Threshold nodes will check this new condition using the `balanceOf` method.

To encrypt the plaintext:

```
const NFTBalanceConfig = {
  contractAddress: '0xBC4CA0EdA7647A8aB7C2061c2E118A18a936f13D',
  standardContractType: 'ERC721',
  chain: 5,
  method: 'balanceOf',
  parameters: [':userAddress'],
  returnValueTest: {
    comparator: '>=',
    value: 3,

  },
};
const NFTBalance = new Conditions.Condition(NFTBalanceConfig);

const encrypter = newDeployed.encrypter;

const plaintext = 'this is a secret';
const encryptedMessageKit = encrypter.encryptMessage(
  plaintext,
  new ConditionSet([NFTBalance])
);
```

The resulting `encryptedMessageKit` contains the encrypted data and associated condition(s).

6. Request decryption rights

Finally, we will test the conditional access control service by requesting decryption rights:

```
const decrypter = newDeployed.decrypter;

const conditionContext = conditions.buildContext(web3Provider);
const decryptedMessage = await decrypter.retrieveAndDecrypt(
  [encryptedMessageKit],
  conditionContext
);
```

At decryption time, the requester will be asked to verify their address by signing a message in MetaMask. This is where `conditionContext` comes into play – if the requester's address controls the minimum number (or greater) of the specified NFT, they are eligible to receive the requisite number of decryption fragments. By assembling these fragments, they are able to decrypt and view the plaintext encrypted in the previous step.

Note that the requester does not need to manually sign the next time they seek access to the data, as their client can temporarily cache their signature. Fresh plaintexts encrypted under any conditions involving the same wallet address are automatically accessible to any requester who has signed at least once, provided they still fulfill the (new) conditions, and the cached signature has not expired.

Example applications

The following samples showcase integrations with React-based web apps, and serve as an 'end-to-end' reference for creating conditions-based encryption & decryption:

- nucypher/tdec-sandbox
- nucypher/tdec-nft-example
- nucypher/alpha-leaks-demo

Trust Assumptions

As the only genuinely decentralized access control service on the market, Threshold Access Control may also claim to be the most 'trustless' solution available to Web3 app buidlers – or indeed, available to any developer seeking an unambiguous departure from the exploitative Web 2.0 trust paradigm.

Nevertheless, there is no such thing as a third-party integration that imposes *zero* additional trust burden onto an application's end-users. Rather, when choosing infrastructural technology, it is the nature and texture of the trust burden(s) – or the 'trust assumptions' – that truly matter.

In the case of Threshold Access Control, the trust assumptions are:

non-binary – i.e. not reducible to 'trustless' or 'trustful'.

non-static – i.e. they evolve over time based on in-protocol features and exogeneous phenomena.

tunable – i.e. adopting developers may select from a range of explicit trust assumptions on behalf of their user base, and/or surface that optionality for end-users to choose themselves.

This section serves to explain the implicit and explicit trust assumptions that are baked into Threshold Access Control, for the Mainnet and Proof-of-Concept versions. These assumptions are synthesized and exposed to the vast majority of adopting developers in the form of pre-configured Trust Packages.

Note that many projects in the Web3 infrastructure space seek to obscure this reality by loosely labelling their solution as 'decentralized', but fail to explain precisely what the trust implications are in practice. Beware of ambiguity!

Trust Packages

This page is under construction.

CBD Mainnet Version

Note that the CBD Mainnet version remains under development and is planned for release in Q2 2023. However, for developers considering integrating Conditions-Based Decryption into their application, it is worthwhile familiarizing oneself with the underlying trust model.

Trust Packages

CBD allows adopting developers to pull a range of 'trust levers' in order to satisfy and assuage the trust, risk, cost, redundancy and latency preferences of their end-users. However, correctly selecting individual parameters across each and every trust/risk dimension requires a deep understanding of the underlying mechanisms and cryptology. Moreover, these parameters must be combined into a coherent bundle that, as a whole, aligns with the risk aversion and trust-minimization desiderata of end-users. Hence, to avoid burdening developers with onerous analysis, the Threshold team has constructed a set of pre-configured 'trust packages'. They are introduced and detailed here.

The context for configuring the trust packages follows below.

Cohort-based trust assumptions

The foundation of the Threshold Access Control trust model (and indeed, threshold cryptography in general) is the concept of a *Cohort;* a group of Threshold nodes temporarily employed to either:

1. Collectively generate and manage part of a public key that can be used to encrypt one or more data payloads.
2. Collectively manage a decryption fragment associated with a single data payload and provide the fragment to requesters who fulfill pre-specified conditions.

All Cohorts are parametrized on formation, including the Cohort *size* (`n`) and Cohort *threshold* (`m`). These parameters are the inputs for the following core trust assumptions:

The first is the *orderly threshold* assumption, wherein the protocol relies on a minimum number – the threshold – of node operators within each Cohort to follow the protocol correctly. For example, a 16-of-32 cohort would require at least **16** nodes to be online, responsive and (ideally) run an up-to-date version of CBD software. If any fewer than 16 are online, data requesters will be unable to retrieve decryption fragments.

The second is the *honest threshold* assumption, the protocol's most fundamental form of collusion-resistance – that is, protection against deliberate, unlawful attempts to access private data. In this case, the protocol relies on a minimum number of operators to be 'honest' – i.e. not susceptible to bribery, coercion or other attempts to maliciously collude. This minimum is calculated as the threshold node count (`m`) subtracted from the total cohort size, plus one (`n - m + 1`). Using the same example as before, a 16-of-32 Cohort would require a minimum of 32 - 16 + 1 = **17** honest operators. In other words, if at least 17 individual operators refuse to collude, there is nothing the remaining 15 operators can do, regardless of their war chest or aggregate stake power.

Note that the *orderly threshold* and *honest threshold* assumptions are conceptually similar to the more common *honest majority* assumption. However, they are more flexible than simply requiring those who control 50% of the staked tokens to be honest. Unlike most BFT or pBFT-based protocols, the *honest threshold* can be partially decoupled from the operators' stake weights, depending on the cohort sampling parameters specified by the developer or end-user.

Sampling-based trust assumptions

The *orderly threshold* and *honest threshold* trust assumptions above treat each Cohort as an isolated group, where the chosen parameters (`m-of-n`) determine the group's redundancy, latency and collusion resistance. However, the reality is that each Cohort is selected from a larger sample of Threshold nodes, which is far larger than the typical/optimal size of each cohort – there is not necessarily much overlap between Cohorts.

Therefore, the mechanisms through which nodes are selected to form Cohorts carry their own trust assumptions. More specifically, the *sampling parameters* impact the security and collusion-resistance of a given data sharing flow. Sampling parametrization can be divided into; (1) those relating to frequency and prompting of (re-)sampling, and (2) compositional requirements to form a Cohort, besides the top-level `m` & `n` parameters.

(1) Cohort refresh

Members of a Cohort can be replaced as frequently as is practical, with the main constraint being the bandwidth overhead of key discovery. A more regular re-sampling of Cohort members provides a form of increased collusion-resistance, as an attacker would have less time to discover, contact and bribe/coerce the requisite threshold of operators in order to access sensitive material.

It could be argued that with very frequent Cohort refresh, a wily attacker could simply wait until a subset of operators they control happens to satisfy the threshold, however briefly that may be. However, this attack strategy would be of little value in most contexts, since it would be almost impossible to predict which encrypted message would coincide with the malicious subset-in-waiting. For example, an attacker seeking illicit access to market sensitive information (from a private DAO group chat) would be extraordinarily fortunate to temporarily control the requisite decryption fragments in the exact window that a market-moving message was shared.

The logic which *prompts* the refreshing of Cohort members is also customizable. For example, a group chat application might include the option to mark a conversation as 'sensitive', in which case messages sent within this conversation will be assigned to and managed by fresh Cohorts with a greater frequency (or randomness) than regular messages.

It's worth noting that the refreshing of Cohort members is unavoidable, regardless of the frequency or prompts chosen by a developer or end-user. This is because node operators will naturally wind down their operations and commitment to the network ('unbonding') over time. Although there are staker-facing protocol mechanisms in place to minimize the probability of a disorderly exodus, the proactive refreshing of Cohorts can also mitigate against abandonment of duties, for example by preemptively replacing operators with pending stake withdrawals (i.e. those who have initiated unbonding) with those with a longer-term economic commitment to the network (i.e. no unbonding initiated, or some longer token lock-up period).

(2) Cohort composition beyond m & n

Developers may also toggle certain Cohort characteristics via the sampling mechanism. For example, they might force a percentage of Cohort positions to be filled exclusively by:

- Node operators who have staked a minimum (or maximum) sum of T tokens.
- Specific Ethereum addresses (e.g. nodes operated by the adopting developer themselves, or their partners).
- Node operators who have not initiated stake unbonding in the past X months.
- Node operators who also (or do not) provision service to other Threshold applications, such as tBTCv2.
- Node operators who have a minimum historical availability (for example, as measured via their verifiable tBTCv2 activity).

Population-based trust assumptions

This section is under construction.

The protocol is also permissionless and psuedonymous (Ethereum addresses), which means the total *population* of node operators will expand, contract and change its composition over time, and that change will not be perfectly discernible.

The node population is also non-uniform; rather, it is comprised of a heterogeneous set of operators, ranging from the hobbyist to institutional stakers. Although there is a degree of standardization with respect to requisite machine memory, CPU power, latency and availability, the actual underlying servers are only observable anecdotally – i.e. this cannot yet be verified on-chain. However, this anecdotal, informal insight into the node population can be supplemented and corroborated with conclusions drawn from on-chain activity, such as slashing or unbonding events.

Infrastructure-based trust assumptions

> This section is under construction.

CBD Proof-of-Concept Version

The Proof-of-Concept version exists to help developers familiarize themselves with the SDK, API and other architectural/design choices, ahead of the Mainnet version release in Q2 2023.. It is not intended to be utilized as a trust-minimized service. Hence, the trust assumptions are strictly worse than the Mainnet version. One can consider the following limitations to be appended on top of the Mainnet trust model:

1. In the absence of a DKG component, the encryptor (or 'Alice') retains permanent decryption power, regardless of their condition fulfillment later – e.g. transferring the corresponding NFT out of their wallet.
2. Conditions can technically be 'stripped' from ciphertexts by a sophisticated user, which would enable them to maliciously gain decryption rights by presenting a ciphertext with no attached access conditionality.
3. The node array managing decryption fragments – and validating condition (non/)fulfillment – are running on testnet, are operated primarily by NuCypher team-members, and are not subject to a cryptoeconomic protocol or a requirement to deposit/risk collateral.

Note that these three limitations will be addressed in the CBD Mainnet version. Upgrading from this version to the Mainnet version involves entirely replacing the underlying cryptosystem, which means all data encrypted via the Proof-of-Concept will have to be (downloaded and) encrypted again from plaintext utilizing the new cryptography.

PRE Mainnet Version

This page is under construction.

CBD Advanced Usage

Condition Hierarchies

Conditions can be attached at several steps in the TAC lifecycle, and they have a fixed hierarchy at runtime. This means default Conditions can be specified and subsequently overwritten later on in the process.

Strategy Conditions

Conditions can be attached directly to a Strategy. They have the lowest precedence and are a great place for including defaults or 'fall back' conditions.

```
import { Cohort, Conditions, ConditionSet, Strategy } from '@nucypher/nucypher-ts';

const config = {
  threshold: 3,
  shares: 5,
  porterUri: 'https://porter-tapir.nucypher.community',
};
const newCohort = await Cohort.create(config);

const NFTOwnership = new Conditions.ERC721Ownership({
  contractAddress: '0xBC4CA0EdA7647A8aB7C2061c2E118A18a936f13D',
  chain: 5, // Tapir network uses Görli testnet
  parameters: [5954],
})
const conditions = new ConditionSet([NFTOwnership]);

const newStrategy = Strategy.create(
  newCohort,
  conditions
);
```

All `encrypter` objects that a deployed Strategy produces will automatically have these conditions included. Therefore, all encrypted messages will require these conditions to be satisfied.

Encrypter Conditions

This is the next level of precedence in the hierarchy, where each encrypter object can have its own conditions. Assuming the above strategy has been deployed, we can attach conditions in the following way:

```
const encrypter = deployedStrategy.encrypter;

const newNFTOwnership = new Conditions.ERC721Ownership({
  contractAddress: '0xBC4CA0EdA7647A8aB7C2061c2E118A18a936f13D',
  chain: 5,
  parameters: [5000], // let's change the specific NFT
})

encrypter.conditions = new ConditionSet([newNFTOwnership])
```

This will **overwrite** the Strategy conditions we defined above - only the new Conditions will be evaluated, not both. All messages encrypted with `encrypter` will require `newNFTOwnership` to be satisfied.

Message Conditions

This is the final, and highest priority, Condition type. When encrypting a message, Conditions can be added that apply **only** to this specific encryption. Again, they will overwrite any Conditions specified during Strategy creation or within the encrypter.

```
const NFTBalance = new Conditions.ERC721Balance({
  contractAddress: '0xBC4CA0EdA7647A8aB7C2061c2E118A18a936f13D',
  chain: 5,
});

const plaintext = 'this is a secret';
const encryptedMessageKit = encrypter.encryptMessage(plaintext, new ConditionSet([NFTBalance]
```

Here we've actually made our Condition more relaxed, and only require a non-zero balance with the NFT contract.

Create Security Optionality With Reusable Cohorts

Providing configurable security to end-users is easy with *Reusable Cohorts*. Below we define three cohorts: small, medium, and large.

`smallCohort`, `mediumCohort`, and `largeCohort` can then be reused by multiple Strategies.

```
const smallConfig = {
  threshold: 3,
  shares: 5,
  porterUri: 'https://porter-tapir.nucypher.community',
};
const mediumConfig = {
  threshold: 11,
  shares: 20,
  porterUri: 'https://porter-tapir.nucypher.community',
};
const largeConfig = {
  threshold: 51,
  shares: 100,
  porterUri: 'https://porter-tapir.nucypher.community',
};
const smallCohort = await Cohort.create(smallConfig);
const mediumCohort = await Cohort.create(mediumConfig);
const largeCohort = await Cohort.create(largeConfig);
```

Implementing Revocation via Smart Contract

It is possible to implement *Revocation* using Conditions that rely on a function call to a Custom Smart Contract. This allows the handling of revocation to be decentralized and transparent. Here is an example of a smart contract (not suitable for production):

```
pragma solidity 0.8.7;

contract Revocation {

    mapping(address => bool) public isRevoked;

    function revoke(address user) public {
        isRevoked[user] = true;
    }

    function unRevoke(address user) public {
        isRevoked[user] = false;
    }
}
```

And the associated Condition:

```
const revocationCondition = {
  contractAddress: 'DEPLOYED_CONTRACT_ADDRESS',
  method: 'isRevoked',
  parameters: [':userAddress'],
  functionAbi: {
    inputs: [
      {
        internalType: 'address',
        name: '',
        type: 'address',
      },
    ],
    name: 'isRevoked',
    outputs: [
      {
        internalType: 'bool',
        name: '',
        type: 'bool',
      },
    ],
    stateMutability: 'view',
    type: 'function',
  },
  chain: 'ethereum',

  returnValueTest: {
    comparator: '==',
    value: false,
  },
};
```

The condition we have defined calls the `isRevoked` function of the smart contract and passes the user's address. If the call returns `false` (**not** revoked, i.e. granted), then decryption will occur. If the call returns `true` (**is** revoked), then decryption will fail.

CBD References

Cohort

`Cohort.create`

When creating a new cohort, the configuration **must** include `threshold`, `shares`, and `porterUri`. We also make available the parameters `include` and `exclude` which can be used to filter particular Nodes.

```
const config = {
  threshold: 3,
  shares: 5,
  porterUri: 'https://porter-ibex.nucypher.community',
};
const newCohort = await Cohort.create(config, (include = []), (exclude = []));
```

`Cohort.toJSON`

A cohort can be serialized to a JSON object so that it can be stored and re-used at a later time.

```
const cohortJSON = newCohort.toJSON();
console.log(cohortJSON);
// {
//     "ursulaAddresses": [
//         "0x5cf1703a1c99a4b42eb056535840e93118177232",
//         "0x7fff551249d223f723557a96a0e1a469c79cc934",
//         "0x9c7c824239d3159327024459ad69bb215859bd25"
//     ],
//     "threshold": 2,
//     "shares": 3,
//     "porterUri": "https://porter-ibex.nucypher.community"
// }
```

`Cohort.fromJSON`

Similarly, we can read in a valid JSON object to build a new Cohort.

```
const importedCohort = Cohort.fromJSON(cohortJSON);
console.log(importedCohort);
// Cohort {
//     ursulaAddresses: [
//     '0x5cf1703a1c99a4b42eb056535840e93118177232',
//     '0x7fff551249d223f723557a96a0e1a469c79cc934',
//     '0x9c7c824239d3159327024459ad69bb215859bd25'
//     ],
//     configuration: {
//     threshold: 2,

//     shares: 3,
//     porterUri: 'https://porter-ibex.nucypher.community'
//     }
// }
```

Conditions

This page focuses on Condition types and composition. To understand how Conditions are added and enforced with respect to runtime, check out the Condition Hierarchies page.

Several distinct categories of access conditions can be specified and combined:

- EVM - on-chain state, eg NFT ownership, ETH balance, tx status, contract function call
- RPC - ethereum RPC calls as defined in the Official API
- Timelock - time-based conditions, eg Block Height

We provide many helper objects to streamline the creation of common conditions. An expressive API also allows much more granular control of conditions, and we will provide examples of both methods wherever possible.

`Conditions.ERC721Ownership`

`Conditions.ERC721Ownership` is a shortcut for building conditions that test for ownership of a specific ERC721 token (NFT):

```
const NFTOwnership = new Conditions.ERC721Ownership({
  contractAddress: '0xBC4CA0EdA7647A8aB7C2061c2E118A18a936f13D',
  parameters: [5954],
});
```

If we want to be more verbose we can use `Conditions.Condition`. The above and below examples are completely equivalent:

```
const NFTOwnershipConfig = {
  contractAddress: '0xBC4CA0EdA7647A8aB7C2061c2E118A18a936f13D',
  standardContractType: 'ERC721',
  chain: 1,
  method: 'ownerOf',
  parameters: [5954],
  returnValueTest: {
    comparator: '==',
    value: ':userAddress',
  },
};
const NFTOwnership = new Conditions.Condition(NFTOwnershipConfig);
```

`Conditions.ERC721Balance`

`Conditions.ERC721Balance` is a shortcut for building conditions that test for ownership of at least one ERC721 token (NFT) within a collection.

```
const NFTBalance = new Conditions.ERC721Balance({
  contractAddress: '0xBC4CA0EdA7647A8aB7C2061c2E118A18a936f13D',
});
```

Alternatively:

```
const NFTBalanceConfig = {
  contractAddress: '0xBC4CA0EdA7647A8aB7C2061c2E118A18a936f13D',
  standardContractType: 'ERC721',
  chain: 5,
  method: 'balanceOf',
  parameters: [':userAddress'],
  returnValueTest: {
    comparator: '>',
    value: 0,
  },
};

const NFTBalance = new Conditions.Condition(NFTBalanceConfig);
```

`Conditions.TimelockCondition`

`Conditions.TimelockCondition` is a shortcut for building conditions that test against block height.

```
const timelock = Conditions.TimelockCondition({
  returnValueTest: {
    comparator: '>',
    value: 100,
  },
});
```

or:

```
const timelockConfig = {
  contractAddress: '',
  standardContractType: '',
  chain: 5,
  method: 'timelock',
  returnValueTest: {
    comparator: '>',
    value: 100,
  },
};
const timelock = Conditions.Condition(timelockConfig);
```

`Conditions.RpcCondition`

`Conditions.RpcCondition` is a shortcut for building conditions that test against standard RPC calls.

```
const rpc = new Conditions.RpcCondition({
  chain: 1,
  method: 'eth_getBalance',
  parameters: [
      ":userAddress",
      "latest"
    ],
  returnValueTest: {
    comparator: ">=",
    value: 10000000000000
  }
});
```

or:

```
const rpcConfig = {
  contractAddress: '',
  standardContractType: '',
  chain: 1,
  method: 'eth_getBalance',
  parameters: [':userAddress', 'latest'],
  returnValueTest: {
    comparator: '>=',
    value: 10000000000000,
  },
};
const rpc = Conditions.Condition(rpcConfig);
```

`Conditions.Condition`

`Conditions.Condition` provides full control over the configuration of a Condition. It takes parameters:

- `contractAddress` is the public address of the contract we'd like to query.
- `standardContractType` can take values from `ERC20` and `ERC721`. Alternatively, an ABI can be passed through if a non standard contract is being used.
- `functionAbi` is the ABI of the function we'd like to call. This is optional if the contract is a standard `ERC20`, `ERC721` or `ERC1155`.
- `chain` - currently only `ethereum` is supported, please Contact Us if you require non-Ethereum based conditions.
- `method` the contract method that will be called.
- `parameters` are the parameters that will be passed to the contract's `method`.
- `returnValueTest` defines how the return value of the contract call should be evaluated.

Non Zero balance of ERC20 Token

Here we're checking whether the user owns any `T` Threshold Network token:

```
const ERC20Conditions = {
  contractAddress: '0xCdF7028ceAB81fA0C6971208e83fa7872994beE5',
  standardContractType: 'ERC20',
  chain: 5,
  method: 'balanceOf',
  parameters: [':userAddress'],
  returnValueTest: {
    comparator: '>',
    value: 0,
  },
};
```

Function call of nonstandard Contract

In this example, we will check that the user is able to vote in the `T` Threshold DAO. The Threshold staking contract is located at `0x01B67b1194C75264d06F808A921228a95C765dd7`. The function we wish to call is `getVotes` which takes an `address` as its parameter. We need to provide the `contractAddress`, `functionName`, `functionParams`, and `functionAbi` when defining the condition.

```javascript
const customABICondition = {
  contractAddress: '0x01B67b1194C75264d06F808A921228a95C765dd7',
  method: 'getVotes',
  parameters: [':userAddress'],
  functionAbi: {
    inputs: [
      {
        internalType: 'address',
        name: 'account',
        type: 'address',
      },

    ],
    name: 'getVotes',
    outputs: [
      {
        internalType: 'uint96',
        name: '',
        type: 'uint96',
      },
    ],
    stateMutability: 'view',
    type: 'function',
  },
  chain: 1,
  returnValueTest: {
    comparator: '>',
    value: 0,
  },
};
```

Condition Set

Conditions can be combined into Condition Sets using `AND` and `OR` operators.

The below example shows how to authenticate that a requester owns an NFT in one of two different collections.

```
const genuineUndead = new Conditions.ERC721Balance({
  contractAddress: '0x209e639a0EC166Ac7a1A4bA41968fa967dB30221',
});
const gnomePals = new Conditions.ERC721Balance({
  contractAddress: '0x5dB11d7356aa4C0E85Aa5b255eC2B5F81De6d4dA',
});
const NFTConditionSet = new ConditionSet([
  genuineUndead,
  Operator.OR(),
  gnomePals,
]);
```

If we wanted to store this Condition Set for later use we could export it to JSON:

```
const NFTConditionSetJSON = NFTConditionSet.toJSON();
console.log(NFTConditionSetJSON);
// [
//     {
//         "chain": "ethereum",
//         "method": "balanceOf",
//         "parameters": [
//             ":userAddress"
//         ],
//         "standardContractType": "ERC721",
//         "returnValueTest": {
//             "comparator": ">",
//             "value": 0
//         },
//         "contractAddress": "0x209e639a0EC166Ac7a1A4bA41968fa967dB30221"
//     },
//     {
//         "operator": "or"
//     },
//     {
//         "chain": "ethereum",
//         "method": "balanceOf",
//         "parameters": [
//             ":userAddress"
//         ],
//         "standardContractType": "ERC721",
//         "returnValueTest": {
//             "comparator": ">",
//             "value": 0
//         },
//         "contractAddress": "0x5dB11d7356aa4C0E85Aa5b255eC2B5F81De6d4dA"
//     }
// ]
```

And then easily import:

```
const newConditionSet = ConditionSet.fromJSON(NFTConditionSetJSON);
```

Strategy

A Strategy combines all possible configuration parameters for using CBD. It takes the following parameters:

- `cohort` - a `Cohort` object
- `conditionSet?` - an optional `ConditionSet`. If used, all encryptions made via this strategy have a default Condition Set assigned
- `aliceSecretKey?` - an optional Secret Key for the encrypter
- `bobSecretKey?` - an optional SecretKey for decrypter

If the optional secret keys are not provided, new ones will be generated instead.

Create a Strategy

Assuming we have a Cohort already defined, we can construct a Strategy:

```
import { Cohort, Strategy } from '@nucypher/nucypher-ts';

const config = {
  threshold: 3,
  shares: 5,
  porterUri: 'https://porter-ibex.nucypher.community',
};
const newCohort = await Cohort.create(config);

const newStrategy = Strategy.create(
  newCohort
);
```

Deploy a Strategy

Before we can encrypt/decrypt, the Threshold network needs to be made aware of our Strategy. We do this by deploying:

```
import detectEthereumProvider from '@metamask/detect-provider';
import providers from 'ethers';

const MMprovider = await detectEthereumProvider();
const mumbai = providers.providers.getNetwork(80001);

if (MMprovider) {
  const web3Provider = new providers.providers.Web3Provider(
    MMprovider,
    mumbai
  );
  const newDeployed = await newStrategy.deploy('test', web3Provider);
}
```

`Strategy.deploy` takes 2 parameters:

- `label` - this is a string that the network uses to identify the strategy
- `provider` - deploying a Strategy requires writing to the smart contract, so a connection to a wallet is required via a Web3 provider

Deploying a strategy returns a new `DeployedStrategy` object. This object grants us access to the `encrypter` and `decrypter` which can then be used throughout an application.

```
const encrypter = newDeployed.encrypter;
const decrypter = newDeployed.decrypter;

const plaintext = 'this is a secret';
const encryptedMessageKit = encrypter.encryptMessage(plaintext);

const decryptedMessage = await decrypter.retrieveAndDecrypt([
  encryptedMessageKit,
]);
```

Import and Export Strategies

Strategies can be exported allowing them to be reused easily. The syntax is the same whether the strategy has been deployed or not.

```
import { DeployedStrategy } from '@nucypher/nucypher-ts';

const configJSON = newDeployed.toJSON();
console.log(configJSON);
/*
LARGE JSON OBJECT
*/
const importedStrategy = DeployedStrategy.fromJSON(configJSON);
```

Similarly, we can import and export the decrypter objects to JSON. This allows us to rebuild the decrypter on a client facing application:

```
import { DeployedStrategy } from '@nucypher/nucypher-ts';

const decrypter = newDeployed.decrypter;
const decrypterJSON = decrypter.toJSON();
// save this JSON are send it over a side channel to a client facing app

// on the client app
import { tDecDecrypter } from '@nucypher/nucypher-ts';
const newDecrypter = tDecDecrypter.fromJSON(decrypterJSON);
```

Get Started (PRE Mainnet)

This page is under construction. In the meantime use the Read-The-Docs and Github Documentation linked below.

External documentation and examples
https://github.com/nucypher/nucypher-ts
https://github.com/nucypher/nucypher-ts-demo
https://docs.nucypher.com/en/latest/application_development/getting_started.html#

Contribution Guide

Download, install, build, and test with:

```
git clone https://github.com/nucypher/nucypher-ts
cd nucypher-ts
yarn install
yarn build
yarn test
```

Development

Install git hooks

```
npx husky install
```

Generate contract typings

```
yarn typechain
```

Prepare a new release

```
yarn run prepare-release
```

Documentation

To launch a local development version of the documentation:

```
cd website
yarn run start
```

This will launch a local server, available at `http://localhost:3000`.

To release a new version of the documentation:

```
yarn run docusaurus docs:version 1.1.0
```

Publishing

Publish a new release on NPM.

Pay attention to the output of these commands and fix your release if needed.

To build and publish a release, run

```
yarn prepare-release
# Or, to publish an alpha release
yarn prepare-release:alpha
```

Follow instructions from the command output to finalize the process.

Extras

Security

Security Policies for Threshold Network

If you identify vulnerabilities with *any* Threshold Network code, please email security@threshold.network with relevant information to your findings. We will work with researchers to coordinate vulnerability disclosure between our stakers, partners, and users to ensure the successful mitigation of vulnerabilities.

Throughout the reporting process, we expect researchers to honor an embargo period that may vary depending on the severity of the disclosure. This ensures that we have the opportunity to fix any issues, identify further issues (if any), and inform our users.

Sometimes vulnerabilities are more sensitive in nature and require extra precautions. We are happy to work together to use a more secure medium, such as Signal. Email security@threshold.network and we will coordinate a communication channel that we're both comfortable with.

A great place to begin your research is by working on our testnet. Please see our documentation to get started. We ask that you please respect network machines and their owners. If you find a vulnerability that you suspect has given you access to a machine against the owner's permission, stop what you're doing and immediately email security@threshold.network.

Contribution

How to make edits

Refer to the following documentation here: https://docs.gitbook.com/

To contribute to Threshold documentation, the best place to start is to join the Discord. You can ask questions there and help contribute to documentation. Join here: http://discord.gg/threshold

Contract Addresses

Click here to see the addresses of contracts deployed on Ethereum Mainnet.

Click here to see the addresses of contracts deployed on Görli Testnet.

Ethereum Mainnet

Threshold Contracts

Delivered in `@threshold-network/solidity-contracts@1.2.1` package.

Contract	Address
T Token	0xCdF7028ceAB81fA0C6971208e83fa7872994beE
TokenStaking	0x01b67b1194c75264d06f808a921228a95c765dd7
KeepStake	0x10DE37cF84202A20cae61069C617B3Aa874aF8
NU<>T Vending Machine	0x1CCA7E410eE41739792eA0A24e00349Dd2476
KEEP<>T Vending Machine	0xE47c80e8c23f6B4A1aE41c34837a0599D5D16bb

PRE Application Contracts

Contract	Address
PRE Application	0x7E01c9c03FD3737294dbD7630a34845B0F70
SubscriptionManager (Polygon-mainnet)	0xB0194073421192F6Cf38d72c791Be8729721A

Legacy NU Contracts

Contract	Address
NU Token	0x4fE83213D56308330EC302a8BD641f1d0113A
StakingEscrow	0xbbD3C0C794F40c4f993B03F65343aCC6fcfC
WorkLock	0xe9778e69a961e64d3cdbb34cf6778281d34667

Legacy Keep Contracts

Delivered in `@keep-network/keep-core@1.8.0` package.

Contract	Address
KEEP Token	0x85Eee30c52B0b379b046Fb0F85F4f3Dc3009a
TokenStaking	0x1293a54e160D1cd7075487898d65266081A15

TBTC Application Contracts

Delivered in `@keep-network/random-beacon@2.0.0` and `@keep-network/ecdsa@2.0.0` packages.

Contract	Address
Bridge	0x5e4861a80B55f035D899f66772117F00FA0E8
RandomBeacon	0x5499f54b4A1CB4816eefCf78962040461be3D
TBTC	0x18084fbA666a33d37592fA2633fD49a74DD93
TBTCVault	0x9C070027cdC9dc8F82416B2e5314E11DFb4F E3CD
VendingMachine	0x6590DFF6abEd7C077839E8227A4f12Ec90E6
WalletRegistry	0x46d52E41C2F300BC82217Ce22b920c349952

Görli Testnet

Threshold Contracts

Delivered in `@threshold-network/solidity-contracts@1.3.0-goerli.0` NPM package.

Contract	Address
T Token	0x3f16380656cAE45D3f80D8833682d2b606eD094
TokenStaking	0x1da5d88C26EA4f87b5e09C3452eE2384Ee20DC
KeepStake	0xC76796B031232976cb17521342B0d12283588E
NU<>T Vending Machine	0x8D4Ccc04D17d055b88fbaB8634C7F0611d8Aa9
KEEP<>T Vending Machine	0x8d1EDfee0510537E1081a28Fa4A94769Cfa3969

Legacy NU Contracts

Contract	Address
NU Token	0x26bDc8Cb2D4F55Af3B43568069b65B435bC
StakingEscrow (stub)	0xd696d5a9b083959587F30e487038529a876b0

Legacy Keep Contracts

Delivered in `@keep-network/keep-core@1.8.1-goerli.0` NPM package.

Contract	Address
KEEP Token	0x22647FfAe391540d584599818CA22fdF18890
TokenStaking	0x73A63e2Be2D911dc7eFAc189Bfdf48FbB6532

TBTC Application Contracts

Delivered in `@keep-network/random-beacon@2.1.0-goerli.6`, `@keep-network/ecdsa@2.1.0-goerli.4` and `@keep-network/tbtc-v2@1.0.3-goerli.0` NPM packages.

Contract	Address
Bridge	0x0Cad3257C4B7ec6de1f6926Fbf5714255a663
RandomBeacon	0xF177CfA720ceC42841c04A458f6c68e1243C1
TBTC	0x679874fBE6D4E7Cc54A59e315FF1eB26668
TBCTVault	0x65eB0562FCe858f8328858c76E689aBedB78
VendingMachine	0x36B7383077a2CEeFd53e796512760a1888cE
WalletRegistry	0x5b1ebF8008097Ac3EF6785220bFb9ecA2B1a

Glossary

Threshold Network Lingo

Term	Definition
Threshold Cryptography	Threshold cryptography is a revolutionary technology that distributes sensitive operations across multiple independent entities – like nodes a network – and requires a threshold, or minimum number of those entities to cooperate for the operation to be successful.
Staking Provider	An ethereum address of a party authorized to operate in the network on behalf of a given owner The staking provider handles the everyday operations on the delegated stake without actuall owning the staked tokens. A staking provider can not simply transfer away delegated tokens, however, it should be noted that the staking provider’s misbehavior may result in slashing tokens and thus the entire staked amount is indee at stake.
Authorizer	An ethereum address appointed by an owner to authorize applications on behalf of the owner. An application must be approved by the authorizer before the staking provider is eligible to participat
Beneficiary	An ethereum address where the rewards for participation are sent, earned by the staking provider, on behalf of the owner. A beneficiary doesn’t sign or publish any protocol-relevant transactions, but any currency or tokens earned b the staking provider will be transferred to the beneficiary.
Delegated stake	An owner’s staked tokens, delegated to the stakir provider by the owner. Delegation enables token owners to have their wallets offline and their stake operated by staking providers on their behalf.
Application	An external smart contract or a set of smart contracts utilizing functionalities offered by Threshold Network. Example applications are: random beacon, proxy re-encryption, and tBTC. Applications authorized for the given staking

provider are eligible to slash the stake delegated t
that staking provider.

Links

Resource	Link
Threshold Website	https://threshold.network
Staking Dashboard	https://dashboard.threshold.network
Testnet Staking Dashbord	https://dashboard.test.threshold.network
PRE Application Dashboard	https://stake.nucypher.network
Github	https://github.com/threshold-network
Discord	https://discord.gg/threshold
Governance	https://forum.threshold.network
Official Blog	https://blog.threshold.network
Twitter	https://twitter.com/thetnetwork

Between Two Keys

An Educational Video Series

Episode 1

Episode 2