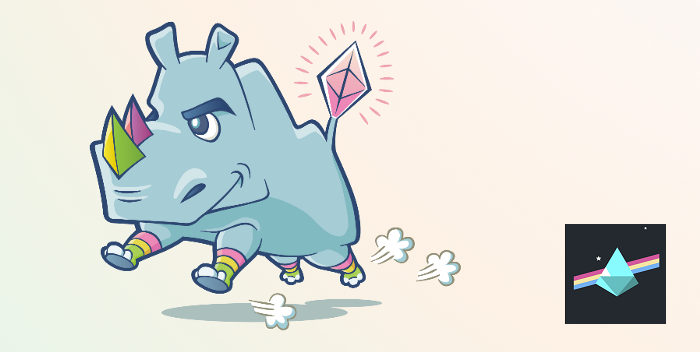
**Guide to Staking on Ethereum 2.0 (Ubuntu/Pyrmont/Prysm)**

[Somer Esat](https://someresat.medium.com/?source=post_page-----a10b5129c7e3--------------------------------)

[Somer Esat](https://someresat.medium.com/?source=post_page-----a10b5129c7e3--------------------------------)

[Nov 18·28 min read](https://someresat.medium.com/guide-to-staking-on-ethereum-2-0-ubuntu-pyrmont-prysm-a10b5129c7e3?source=post_page-----a10b5129c7e3--------------------------------)

Image for post



This is a step-by-step guide to staking on Ethereum 2.0 via the Pyrmont multi-client testnet. It is based on the following technologies:

* [Ubuntu](https://ubuntu.com/) v20.04 (LTS) x64 server
* [Go Ethereum](https://geth.ethereum.org/docs/) Node ([code branch](https://github.com/ethereum/go-ethereum))
* [Prysmatic Labs](https://prysmaticlabs.com/) Ethereum 2.0 client — Prysm ([code branch](https://github.com/prysmaticlabs/prysm))
* Official multi-client testnet public network, [Pyrmont](https://github.com/protolambda/pyrmont)
* [MetaMask](https://metamask.io/) crypto wallet browser extension
* [Prometheus](https://prometheus.io/) metrics
* [Grafana](https://grafana.com/) dashboard

This guide includes instructions on how to:

* Configure a newly running Ubuntu server instance
* Configure and run an Ethereum 1.0 node as a service
* Generate a Prysm wallet and import Pyrmont validator account keys
* Compile and configure the Prysmatic Labs beacon chain and validator client software for Ethereum 2.0, Phase 0 (Pyrmont testnet) and run them as a service
* Install and configure Prometheus metrics and set up a Grafana dashboard

Warnings

This guide is for the **Pyrmont testnet**. DO NOT, under any circumstances, send mainnet ETH to this testnet. You will lose it.

This guide SHOULD NOT be used for connecting to the Eth2 mainnet. An Eth2 mainnet specific guide is currently under development.

Acknowledgements and Disclaimer

This guide is based on information I pulled together from various on-line resources and this guide wouldn’t exist without them. Thank you, all!

Thanks to the folks on the [Prysmatic](https://discord.gg/VaQcHq76yJ) and [EthStaker](https://discord.gg/7z8wzehjrJ) discords for their help and review.

Special thanks to the client team and the EF researchers. Your tireless efforts over the past few years have brought us to the cusp of an incredible moment in history — the launch of Ethereum 2.0.

This is for educational purposes only. I’m not an expert in any of the technologies listed in this guide. The accuracy of this guide is not guaranteed and I am not responsible for any damages or losses incurred by following this guide.

Feedback is always welcome!

Support

This stuff can be tricky. If you need help there are two great resources you can reach out to (besides me):

* EthStaker community on [Reddit](https://www.reddit.com/r/ethstaker/) or [Discord](https://discord.gg/7z8wzehjrJ). Knowledgeable and friendly community passionate about staking on Ethereum 2.0.
* Prysm client team [Discord](https://discord.gg/VaQcHq76yJ). The client software engineering team. Experts on Prysm and its usage.

Prerequisites

This guide assumes some knowledge of Ethereum, ETH, staking, Linux, and MetaMask.

This guide also requires the following are installed and running before getting started:

* [Ubuntu server v20.04 (LTS) amd64](https://ubuntu.com/tutorials/install-ubuntu-server#1-overview) or newer, installed and running on a local computer on your network or in the cloud (AWS, Digital Ocean, Microsoft Azure, etc.). A local computer is recommended for greater decentralization — if the cloud provider goes down then all nodes hosted with that provider go down.
* [MetaMask crypto wallet browser extension](https://metamask.zendesk.com/hc/en-us/articles/360015489531-Getting-Started-With-MetaMask-Part-1-), installed and configured. A computer with a desktop (Mac, Windows, Linux) and a browser (Safari, Brave, FireFox, etc.) is required.

Note for Raspberry Pi Users

I haven’t tested this guide on a Rpi. If you want to try, just swap out the software listed below for the ARM version, where available. No guarantee it will work!

Requirements

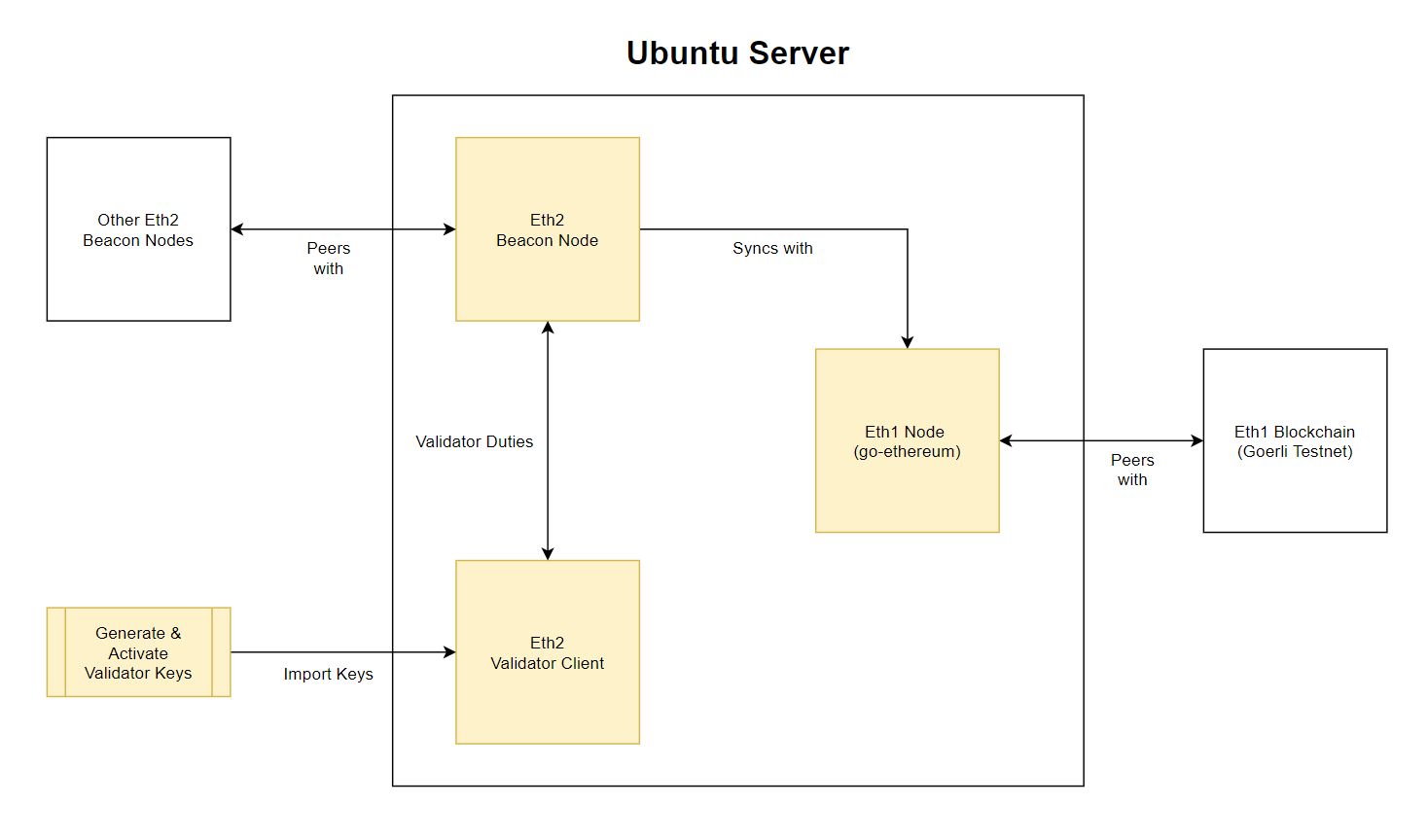
* Ubuntu server instance. I used v20.04 (LTS) amd64 server VM.
* MetaMask crypto wallet browser extension, installed and configured.
* Hardware [recommended requirements](https://docs.prylabs.network/docs/install/install-with-bazel) to run Prysm software:  
  - Processor: Intel Core i7–4770 or AMD FX-8310 or better  
  - Memory: 16GB RAM  
  - Storage: 100GB available space SSD (Prysm client only)

NOTE: Hardware requirements are a broad topic. In general a relatively modern CPU, 16GB RAM, a SSD of at least 1TB, and a stable internet connection with sufficient download speed and monthly data allowance are *likely*required for good staking performance.

Overview

This is a long and detailed guide. Here’s a super-simplified diagram to help you conceptualize what we are going to do. The yellow boxes are the parts this guide mostly covers.

Image for post



The conceptual flow is:

* Set up a Eth1 node and sync it with the Eth1 Göerli testnet
* Configure Beacon Node and sync it with the Eth1 Node
* Generate and activate validator keys
* Configure the Validator Client
* The Beacon Node makes the magic happen (blocks, attestations, slashings) with the help of the validator (signing).

Let’s get started!

Step 0 — Connect to the Server

Using a SSH client, connect to your Ubuntu server. The root user account on Ubuntu server is normally disabled by default, however some cloud providers enable it. If you are logged in as root then let’s create a user-level account with admin privileges instead, because using the root user to log in is [risky](https://askubuntu.com/questions/16178/why-is-it-bad-to-log-in-as-root).

*NOTE: If you are not logged in as root then skip this and go to Step 1.*

# adduser <yourusername>

You will asked to create a password and some other information.

Grant admin rights to the new user by adding it to the sudo group.

# usermod -aG sudo <yourusername>

When you log in as <yourusername> you can type sudo before commands to perform actions with superuser privileges.

Optional: If you used [SSH keys](https://jumpcloud.com/blog/what-are-ssh-keys) to connect to your Ubuntu instance via the root user you will need to associate the new user with the root user’s SSH key data.

# rsync --archive --chown=<yourusername>:<yourusername> ~/.ssh /home/<yourusername>

Finally, log out of root and log in as <yourusername>.

Step 1 — Update the Server

Make sure your system is up to date with the latest software and security updates.

$ sudo apt update && sudo apt upgrade  
$ sudo apt dist-upgrade && sudo apt autoremove

Step 2 — Secure the Server

Security is important. This is not a comprehensive security guide, just some basic settings: a firewall and a user account.

Configure the firewall

Ubuntu 20.04 servers can use the default [UFW firewall](https://help.ubuntu.com/community/UFW) to restrict inbound traffic to the server. Before we enable it we need to allow inbound traffic for SSH, Go Ethereum, Grafana, and Prysm.

**Allow SSH**Allows connection to the server over SSH. For security reasons we are going to modify the default port of 22 because it is a common attack vector.

*Note: If you would rather not change the default SSH port (not recommended), then include a rule allowing the default SSH port*$ sudo ufw allow 22/tcp*and move onto the “Allow Go Ethereum” section.*

Choose a port number between 1024–49151 and run the following command to make sure your selection is not already in use on the server. If it is (red text), choose a different port.

$ sudo ss -tulpn | grep ':<yourSSHportnumber>'

Update the firewall to allow inbound traffic on <yourSSHportnumber>. SSH requires TCP.

$ sudo ufw allow <yourSSHportnumber>/tcp

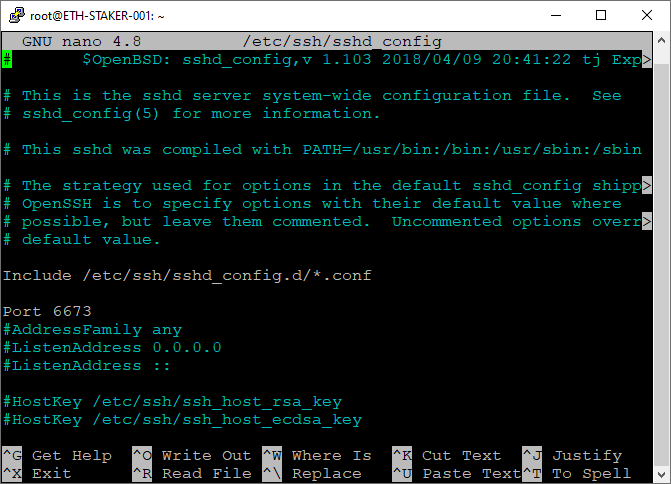
Next change the default SSH port.

$ sudo nano /etc/ssh/sshd\_config

Find the line with # Port 22 or Port 22 and change it to Port <yourSSHportnumber>. Remove the # if it was present.

Check the screen shot below for reference. Your file should look similar to that (but with the port number you chose). Exit and save.

Image for post



Restart the SSH service.

$ sudo systemctl restart ssh

Next time you log in via SSH use <yourSSHportnumber> for the port.

Optional: If you were already using UFW with port 22/TCP allowed then update the firewall to deny inbound traffic on that port. Only do this after you log in using the new SSH port.

$ sudo ufw deny 22/tcp

**Allow Go Ethereum**  
Allows incoming requests from Go Ethereum peers (port 30303/TPC and 30303/UDP). If you’d rather use a node hosted by a 3rd party ([Infura](https://infura.io/), etc.) then skip this step.

*Note: If you are hosting your Ubuntu instance locally your internet router may need to be configured to allow incoming traffic on these ports as well.*

$ sudo ufw allow 30303

**Allow Prysm**Allows P2P connections with peers for actions on the beacon node. Ports 13000/TCP and 12000/UDP are [listed](https://docs.prylabs.network/docs/prysm-usage/p2p-host-ip/#incoming-p2p-connection-prerequisites) as defaults by Prysmatic Labs.

*Note: If you are hosting your Ubuntu instance locally your internet router and/or firewall will need to be configured to allow incoming traffic on these ports as well.*

$ sudo ufw allow 13000/tcp  
$ sudo ufw allow 12000/udp

**Allow Grafana**Allows incoming requests to the Grafana web server (port 3000/TCP).

$ sudo ufw allow 3000/tcp

**Allow Prometheus (Optional)**  
If you want direct access to the Prometheus data service you can open up port 9090/TCP as well. This is not necessary if you are solely using Grafana to view the data. I did not open this port.

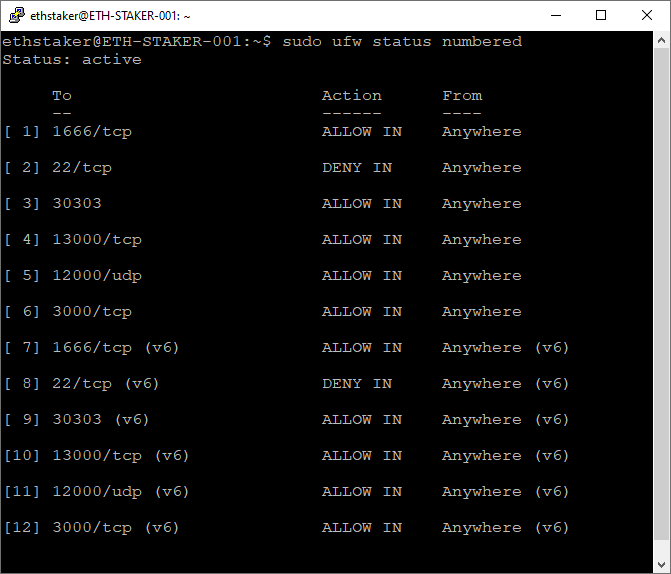
$ sudo ufw allow 9090/tcp

Enable the firewall and check to verify the rules have been correctly configured.

$ sudo ufw enable  
$ sudo ufw status numbered

Output should look something like this. 1666/tcp is the SSH port number I used.

Image for post



Step 3— Install and Run Go Ethereum Node

Install and configure an Ethereum 1.0 node that the beacon-chain will connect to. If you’d rather use a node hosted by a 3rd party ([Infura](https://infura.io/), etc.) then skip this step.

Install Go Ethereum

Go Ethereum recommends using PPA’s (Personal Package Archives).

$ sudo add-apt-repository -y ppa:ethereum/ethereum

Update the packages and install the latest stable version.

$ sudo apt update  
$ sudo apt install geth

Run Go Ethereum as a Background Service

Create an account for the service to run under. This type of account can’t log into the server.

$ sudo useradd --no-create-home --shell /bin/false goeth

Create the data directory for the Eth1 chain. This is required for storing the Eth1 node data. Use the -p option to create the full path.

$ sudo mkdir -p /var/lib/goethereum

Set directory permissions. The goeth account needs permission to modify the data directory.

$ sudo chown -R goeth:goeth /var/lib/goethereum

Create a systemd service file to store the service config. We will use the config file to tell systemd to run the geth process.

$ sudo nano /etc/systemd/system/geth.service

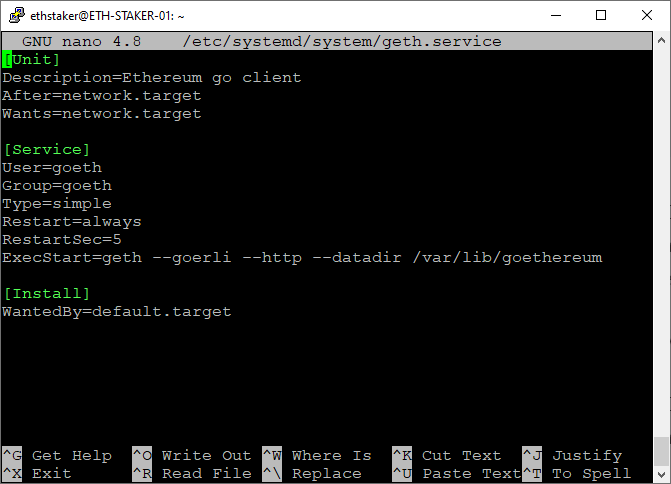
Paste the following service configuration into the file.

[Unit]  
Description=Ethereum go client  
After=network.target   
Wants=network.target[Service]  
User=goeth   
Group=goeth  
Type=simple  
Restart=always  
RestartSec=5  
ExecStart=geth --goerli --http --datadir /var/lib/goethereum[Install]  
WantedBy=default.target

The --goerli flag is used to target the Göerli test network and the --http flag is to expose an endpoint (http://localhost:8545) that the beacon chain will connect to.

Check the screen shot below for reference. Your file should look like that. Exit and save.

Image for post



Reload systemd to reflect the changes.

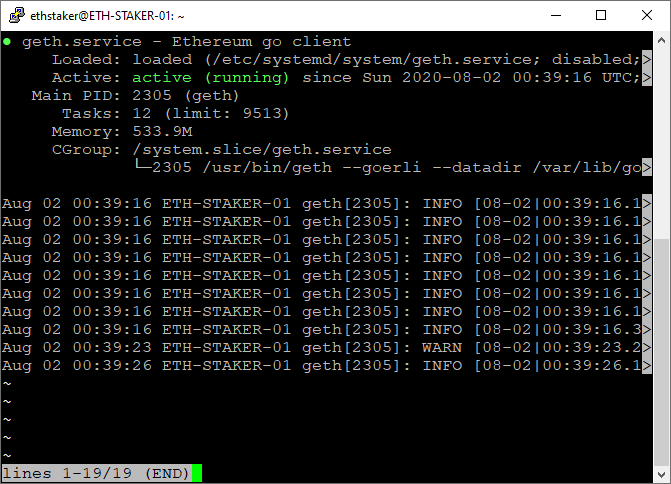
$ sudo systemctl daemon-reload

Start the service and check to make sure it’s running correctly.

$ sudo systemctl start geth  
$ sudo systemctl status geth

Output should look like this.

Image for post



If you did everything right, it should say active (running) in green text. If not then go back and repeat the steps to fix the problem. Press Q to quit.

Enable the geth service to automatically start on reboot.

$ sudo systemctl enable geth

The Go Ethereum node will begin to sync. You can follow the progress by running the journal command. Press Ctrl+C to quit.

$ sudo journalctl -fu geth.service

It may take a while to find peers to sync. If so you can add some peers to help things along. Go [here](https://gist.github.com/rfikki/77081600ddc8432520d3bb3a9f80a493) for the latest list and modify the geth service as follows:

$ sudo systemctl stop geth  
$ sudo nano /etc/systemd/system/geth.service

Modify the ExecStart line and add the --bootnodes flag with a few of the latest peers (comma separated).

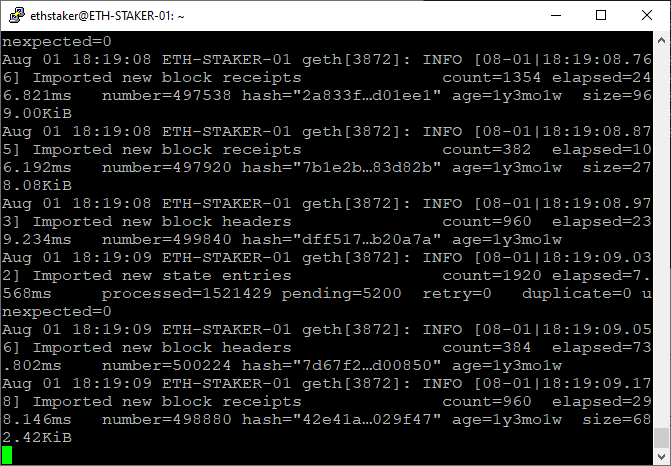
ExecStart=geth --goerli --http --datadir /var/lib/goethereum --bootnodes "enode://46add44b9f13965f7b9875ac6b85f016f341012d84f975377573800a863526f4da19ae2c620ec73d11591fa9510e992ecc03ad0751f53cc02f7c7ed6d55c7291@94.237.54.114:30313,enode://119f66b04772e8d2e9d352b81a15aa49d565590bfc9a80fe732706919f8ccd00a471cf8433e398c55c4862aadb4aadf3a010201483b87e8358951698aa0b6f07@13.250.50.139:30303"

Save the file and exit. Restart the service and observe.

$ sudo systemctl daemon-reload  
$ sudo systemctl start geth  
$ sudo journalctl -fu geth.service

Once it gets started the output should look like this.

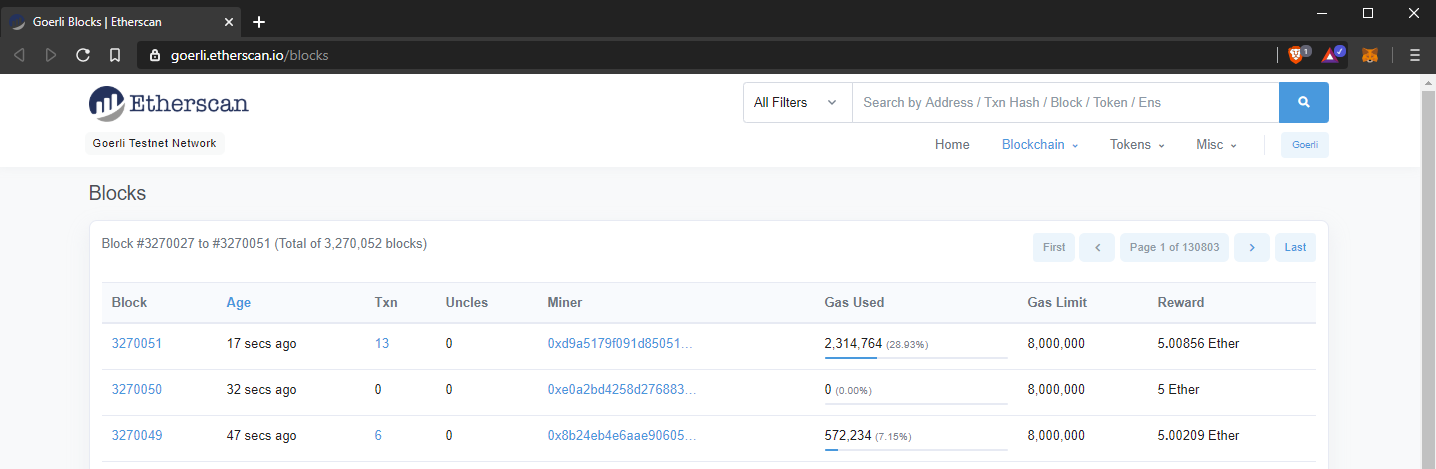
Image for post



You should wait for the node sync to complete before you run the beacon chain. You can see the latest block [here](https://goerli.etherscan.io/blocks).

For example, the screen shot above shows the node is processing block number=498880 and looking at the screen shot below, we can see the latest block number is 3270051. So based on that we still have a while to go before completing the sync.

Image for post



Next we will clone and build the Prysm software (beacon-node and validator). Consider opening a new terminal window here so you can continue to observe the Eth1 node sync.

Step 4— Install Bazel

Bazel is an open-source build tool. We will use it to compile the Prysm software.

[Curl](https://curl.haxx.se/) is required so we can download the Prysm code.

$ sudo apt install curl gnupg

Download and add the Bazel gpg distribution URI as a package source.

$ curl -fsSL https://bazel.build/bazel-release.pub.gpg | gpg --dearmor > bazel.gpg  
$ sudo mv bazel.gpg /etc/apt/trusted.gpg.d/  
$ echo "deb [arch=amd64] https://storage.googleapis.com/bazel-apt stable jdk1.8" | sudo tee /etc/apt/sources.list.d/bazel.list

According to Bazel’s [documentation](https://docs.bazel.build/versions/3.4.0/install-ubuntu.html), the component name “jdk1.8” is kept for legacy reasons only and doesn’t relate to supported or included JDK versions anymore.

Install Bazel. First install the latest version, then install 3.2.0. Prysm currently requires version 3.2.0.

$ sudo apt update && sudo apt install bazel  
$ sudo apt update && sudo apt full-upgrade  
$ sudo apt update && sudo apt install bazel-3.2.0

Install some prerequisites to allow us to compile with the --config=release flag.

$ sudo apt install -y libtinfo5 # Terminal handling  
$ sudo apt-get install -y libssl-dev # OpenSSL   
$ sudo apt-get install -y libgmp-dev # GMP source to build BLS

Step 5— Install and Build Prysm

The Prysm client is made up of two binaries: the beacon-chain and the validator. We will build both here.

First, go [here](https://github.com/prysmaticlabs/prysm/releases) and identify the latest release. It is at the top of the page with the “Latest release” label. For example:

Image for post



*NOTE: We want to pull the latest release and not*master*because it may be unstable.*

We will use the tag v1.0.0-beta.3 from above in the command below to clone that particular release. The --single-branch flag prevents fetching all branches.

$ git clone -b v1.0.0-beta.3 --single-branch https://github.com/prysmaticlabs/prysm.git  
$ cd prysm

Use Bazel Build to compile the beacon chain and validator binaries.

$ bazel build //beacon-chain:beacon-chain --config=release  
$ bazel build //validator:validator --config=release

The Beacon chain takes a while to build the first time around. Good time to get a fresh beverage and re-hydrate. Maybe check out a few of my [other articles](https://someresat.medium.com/).

Building the validator is faster as it is smaller in size and most of the dependencies have already been downloaded and/or built when building the beacon-chain.

If both builds succeed then continue. If not get help on the [Prysm Discord](https://discord.gg/VaQcHq76yJ).

*NOTE: If you need to update the binaries with the latest code changes from Prysmatic Labs follow the instructions at the end of this guide in the****Appendix — Updating Prysm****section.*

Step 6— Configure the Beacon Chain

We will run the beacon chain as a service so if the system restarts the process will automatically start back up again.

Setup Accounts and Directories

Create an account for the service to run under. This type of account can’t log into the server.

$ sudo useradd --no-create-home --shell /bin/false prysm-beaconchain

Create the data directory for the beacon chain. This is required for storing the beacon chain database.

$ sudo mkdir -p /var/lib/prysm/beaconchain

Set directory permissions. The prysm-beaconchain account needs permission to modify the database directory.

$ sudo chown -R prysm-beaconchain:prysm-beaconchain /var/lib/prysm/beaconchain

Copy the Beacon Chain Binary

Next, copy the newly compiled beacon-chain binary to the /usr/local/bin directory. We will run it from there.

*NOTE: You will need to do this step each time you pull/build a new version of the*beacon-chain*binary. See****Appendix — Updating Prysm****at the end of this guide.*

$ cd ~  
$ sudo cp prysm/bazel-bin/beacon-chain/beacon-chain\_/beacon-chain /usr/local/bin

Create and Configure the Service

Create a systemd service file to store the service config.

$ sudo nano /etc/systemd/system/prysm-beaconchain.service

Paste the following into the file.

[Unit]  
Description=Prysm Beaconchain  
Wants=network-online.target  
After=network-online.target[Service]  
Type=simple  
User=prysm-beaconchain  
Group=prysm-beaconchain  
Restart=always  
RestartSec=5  
Environment="ClientIP=$(curl -s v4.ident.me)"  
ExecStart=/bin/bash -c '/usr/local/bin/beacon-chain --pyrmont --p2p-host-ip=${ClientIP} --datadir=/var/lib/prysm/beaconchain --http-web3provider=http://127.0.0.1:8545 --accept-terms-of-use'[Install]  
WantedBy=multi-user.target

The --pyrmont flag is required to indicate we are running against the testnet.

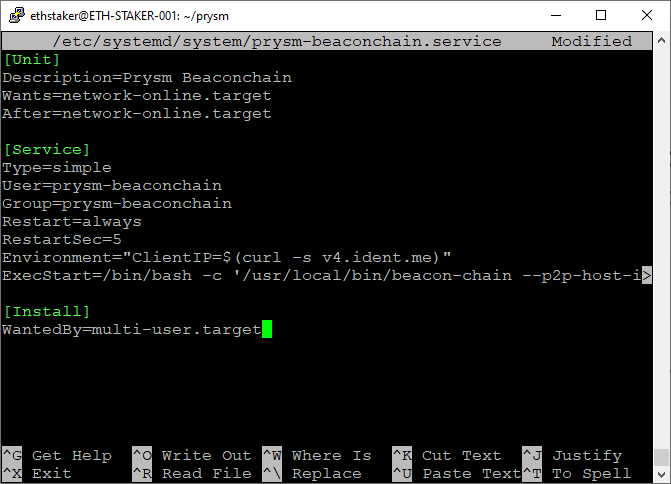
The --p2p-host-ip flag is [recommended](https://docs.prylabs.network/docs/prysm-usage/p2p-host-ip#setting-the---p2p-host-ip-flag) to improve peer networking. We use an Environment variable Environment="ClientIP=$(curl -s v4.ident.me)" to get the client IP address because ExecStart doesn’t allow the call in-line. Using --p2p-host-ip=${ClientIP} is the work-around.

The --http-web3provider flag defines the endpoint of the Eth1 node. If you installed one locally the value is http://127.0.0.1:8545. If you’re using a third party use the external endpoint address (e.g. Infura or Prysmatic’s Eth1 node: https://goerli.prylabs.net).

The --accept-terms-of-use flag is required in order to be able to run the binary as a service. Using this flag indicates acceptance of the Prysm [terms of use](https://github.com/prysmaticlabs/prysm/blob/master/TERMS_OF_SERVICE.md).

Check the screen shot below for reference. Your file should look like that. Exit and save.

Image for post



Reload systemd to reflect the changes.

$ sudo systemctl daemon-reload

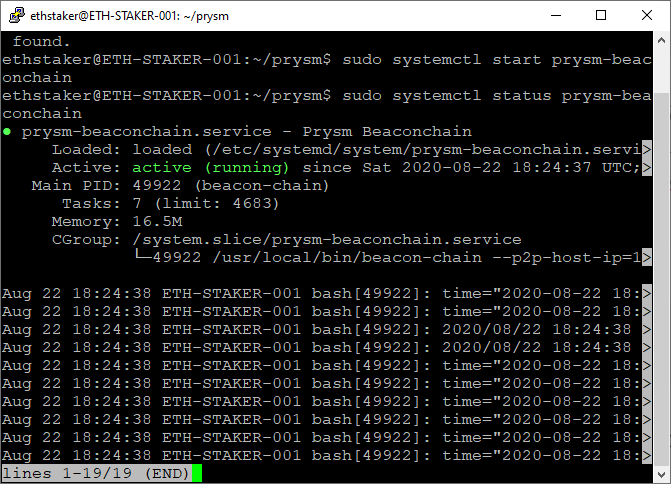
*NOTE: If you are running a local Eth1 node (see Step 3) you should wait until it fully syncs before starting the beaconchain service. Check progress here:*sudo journalctl -fu geth.service

Start the service and check to make sure it’s running correctly.

$ sudo systemctl start prysm-beaconchain  
$ sudo systemctl status prysm-beaconchain

Output should look like this.

Image for post



If you did everything right, it should say active (running) in green text. If not then go back and repeat the steps to fix the problem. Press Q to quit.

Enable the service to automatically start on reboot.

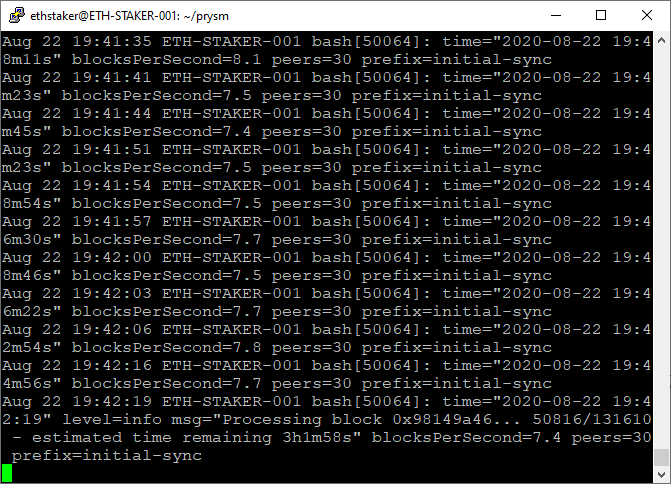
$ sudo systemctl enable prysm-beaconchain

The beacon-chain will begin to sync. It may take several hours for the node to fully sync. You can check the progress by running the journal command. Press Ctrl+C to quit.

$ sudo journalctl -fu prysm-beaconchain.service

The terminal output gives status information that it is processing deposits from the Eth1 chain.

Image for post



Now your beacon chain is running as a service. Congratulations!

Step 7— Complete the Pyrmont On-boarding Process

In order to run a validator on the Eth2.0 Pyrmont testnet we will need to sign up for one or more validator accounts.

*NOTE: If you have already generated your deposit data and submitted your staking deposits you can skip this step. If you generated them elsewhere you will need to copy your validator keys onto the server.*

The steps to sign-up are:

* Get Göerli ETH
* Generate the validator keys. Each key is a validator account
* Fund the validator account(s) with 32 Göerli ETH per account
* Wait for your validator account(s) to become active

Let’s get started.

Get Goerli ETH

1. Go to a computer with the MetaMask browser extension installed.
2. Click on MetaMask and log in.
3. Using the drop-down at the top, select the **Göerli Test Network**.
4. Click on the account name to copy your Göerli Test Network wallet address.
5. Using your address, get Göerli ETH from the [authenticated faucet](https://faucet.goerli.mudit.blog/) or via the #request-goerli-eth channel on the [ethstaker Discord](https://discord.gg/7z8wzehjrJ) using the bot command: !goerliEth <yourwalletaddress>.

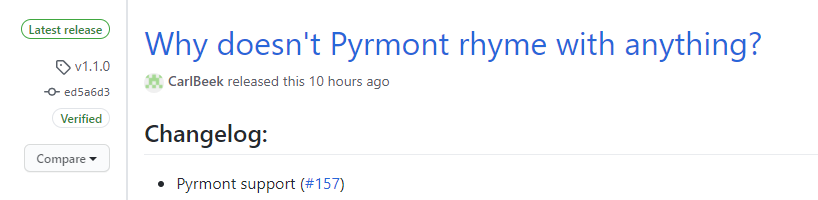
*NOTE: Each validator requires a 32 ETH deposit. You should have sufficient Göerli ETH in your MetaMask wallet to fund each validator. For example, if you want 10 validators you need to have 320 Göerli ETH plus some extra (e.g. 1 Göerli ETH) to pay for the gas fees.*

Generate Validator Keys

Next we will generate the validator keys. The validator client supports multiple validator keys. Each validator key is basically a “validator account” on the Pyrmont testnet.

Go [here](https://github.com/ethereum/eth2.0-deposit-cli/releases/) to get the “Latest release” of the deposit command line interface app.

Image for post



In the assets section copy the link to the Linux version. We will use that link to download it as shown below. Modify the the URL name in the instructions below to match the latest version download link.

$ cd ~  
$ curl -LO https://github.com/ethereum/eth2.0-deposit-cli/releases/download/v1.1.0/eth2deposit-cli-ed5a6d3-linux-amd64.tar.gz

Unpack the tar archive.

$ tar xvf eth2deposit-cli-ed5a6d3-linux-amd64.tar.gz  
$ cd eth2deposit-cli-ed5a6d3-linux-amd64

Clean up by removing the downloaded tar archive file.

$ rm -rf eth2deposit-cli-ed5a6d3-linux-amd64.tar.gz

Run the application to generate the validator keys.

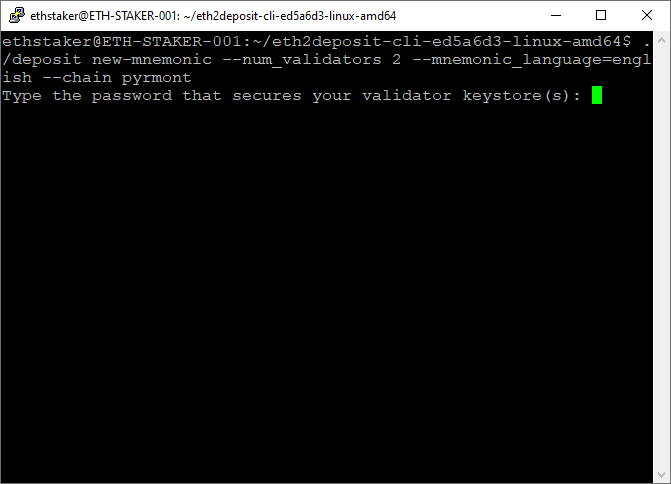
*NOTE: For mainnet this should ideally be done on a fresh machine not connected to the internet to avoid leaking your mnemonic.*

Change <numberofvalidators> to the number of validator keys you want to create. E.g. --num\_validators 5.

$ ./deposit new-mnemonic --num\_validators <numberofvalidators> --mnemonic\_language=english --chain pyrmont

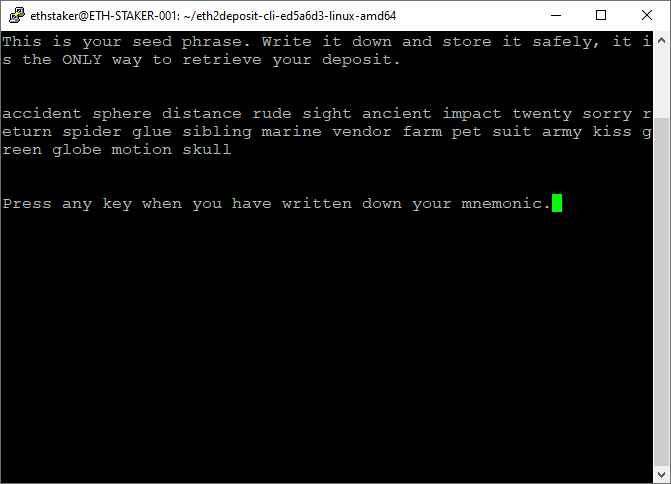
It will ask you to create a wallet password. We will use this to load the validator keys into your client’s validator wallet. **Back it up** somewhere safe.

Image for post



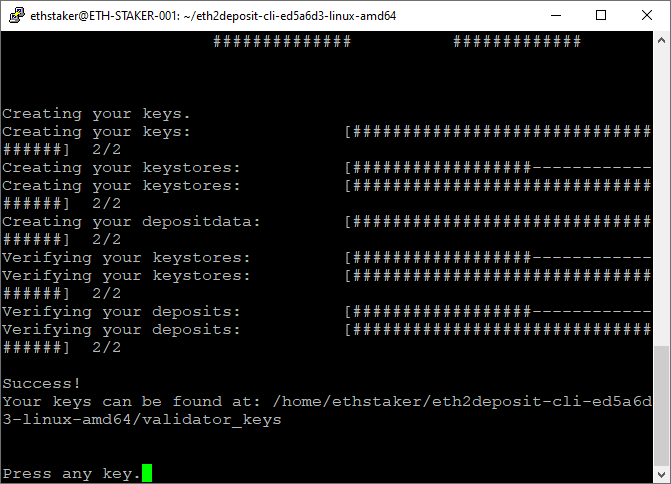
A seed phrase (mnemonic) will be generated. **Back it up**somewhere safe.

Image for post



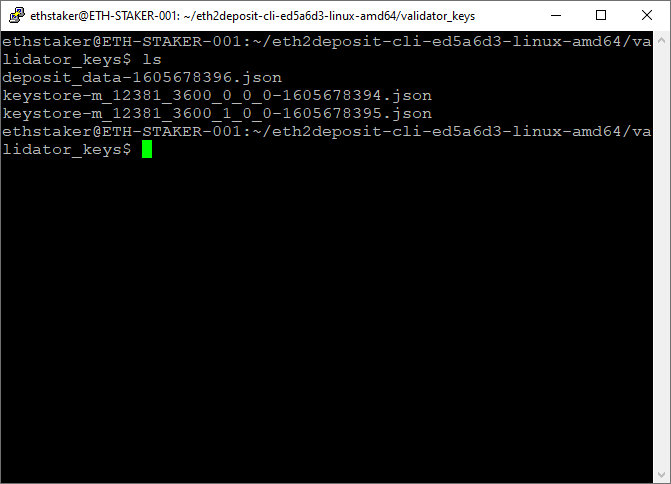
Once you have confirmed your mnemonic your validator keys will be created.

Image for post



The newly created validator keys and deposit data file are created at the specified location.

Image for post



The deposit\_data-[timestamp].json file contains the public keys for the validators and information about the deposit. This file will be used to complete the deposit process in the next step. Since we are on a server we don’t have a web browser so [secure FTP (SFTP)](https://www.maketecheasier.com/use-sftp-transfer-files-linux-servers/) the file over to a computer running MetaMask.

The keystore-m...json files contain the encrypted signing key. There is one keystore-m per validator. These will be used to create the client validator wallet.

Fund the Validator Keys

This step involves depositing the required amount of Göerli ETH to the Pyrmont testnet staking contract. This is done on the Eth2.0 Lauchpad website.

*WARNING: DO NOT send real ETH to the Pyrmont testnet. You will lose your ETH.*

Go here: <https://pyrmont.launchpad.ethereum.org/>

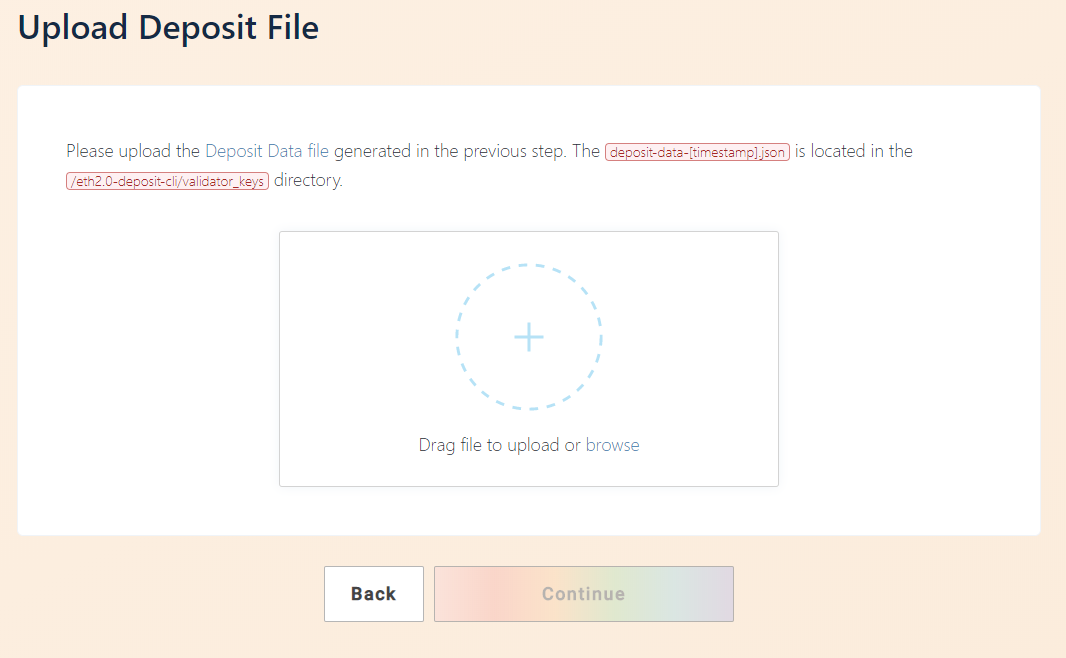
Click through the warning steps then select the number of validators you are going to run. Scroll down and click continue.

Image for post



You will be asked to upload the deposit\_data-[timestamp].json file. This should have been copied over in the previous step. Browse or drag the file and click continue.

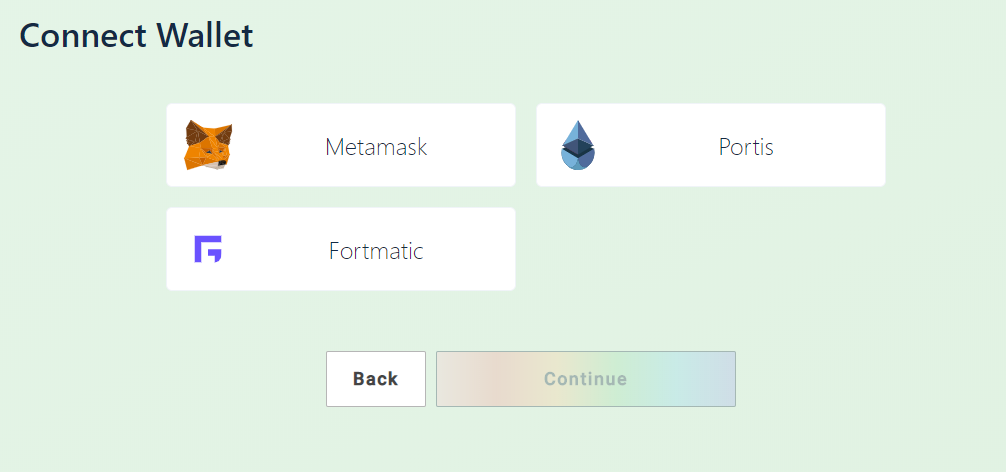
Image for post



Connect your wallet. Choose MetaMask, log in, select the Göerli Test Network and click Continue.

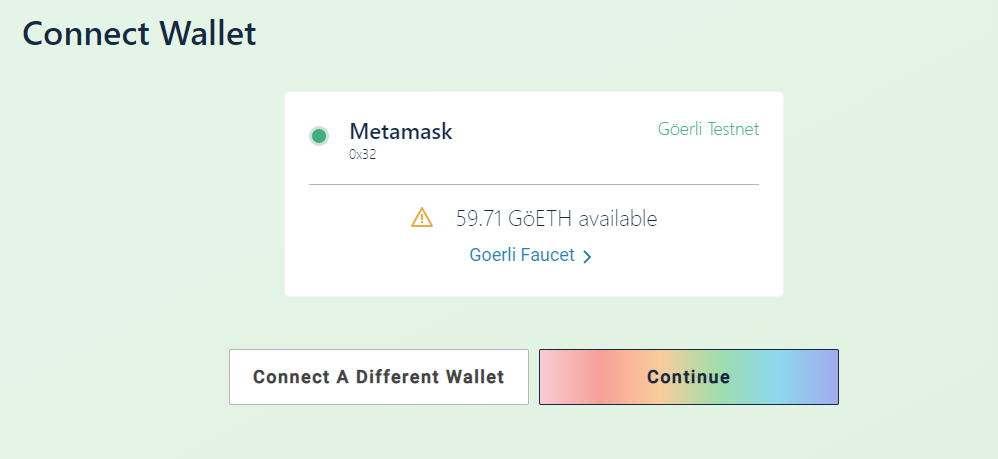
*WARNING: The launchpad should prevent you from using anything other than the Göerli Test Network, but be absolutely 100% sure you have selected the Göerli Test Network in MetaMask. DO NOT sent real ETH to the Pyrmont testnet.*

Image for post



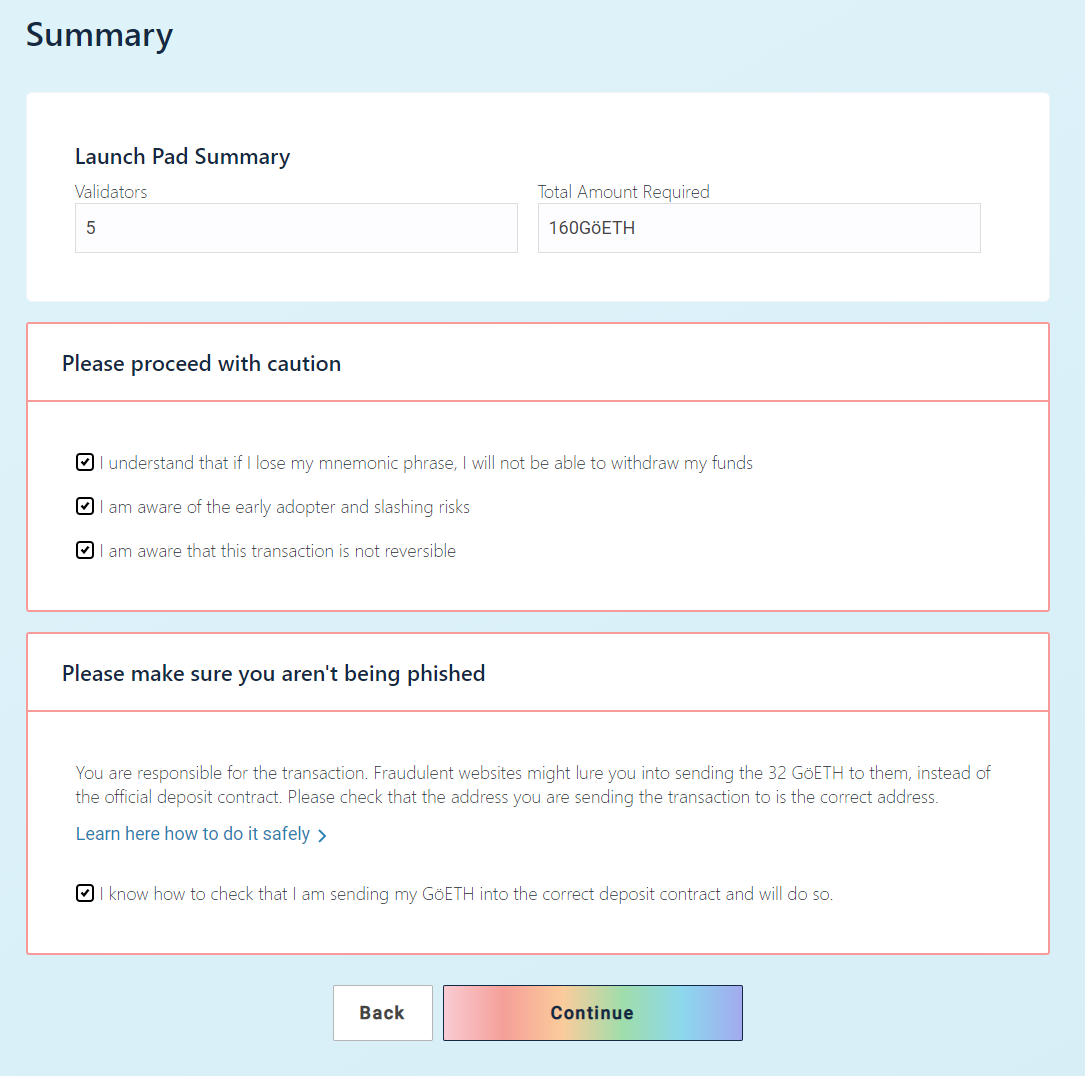
Your MetaMask balance will be displayed. The site will allow you to continue if you have sufficient Göerli ETH balance.

Image for post



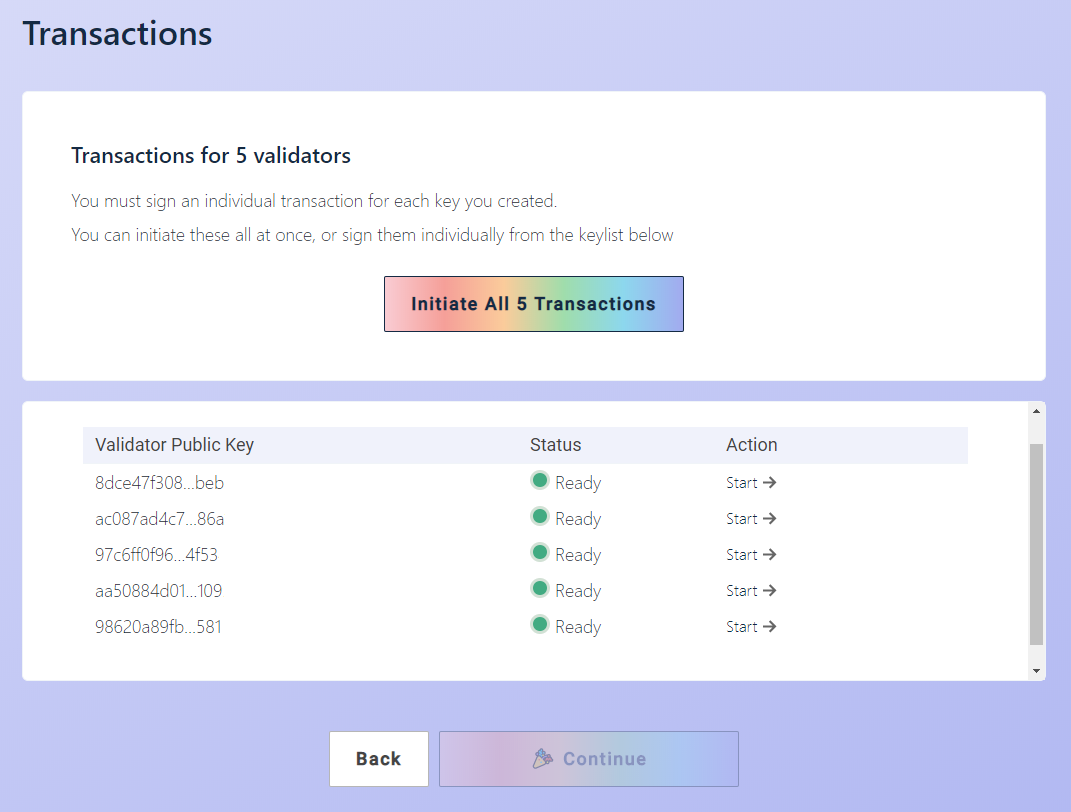
A summary shows the number of validators and total amount of Göerli ETH required. Tick the boxes if you agree and click continue.

Image for post



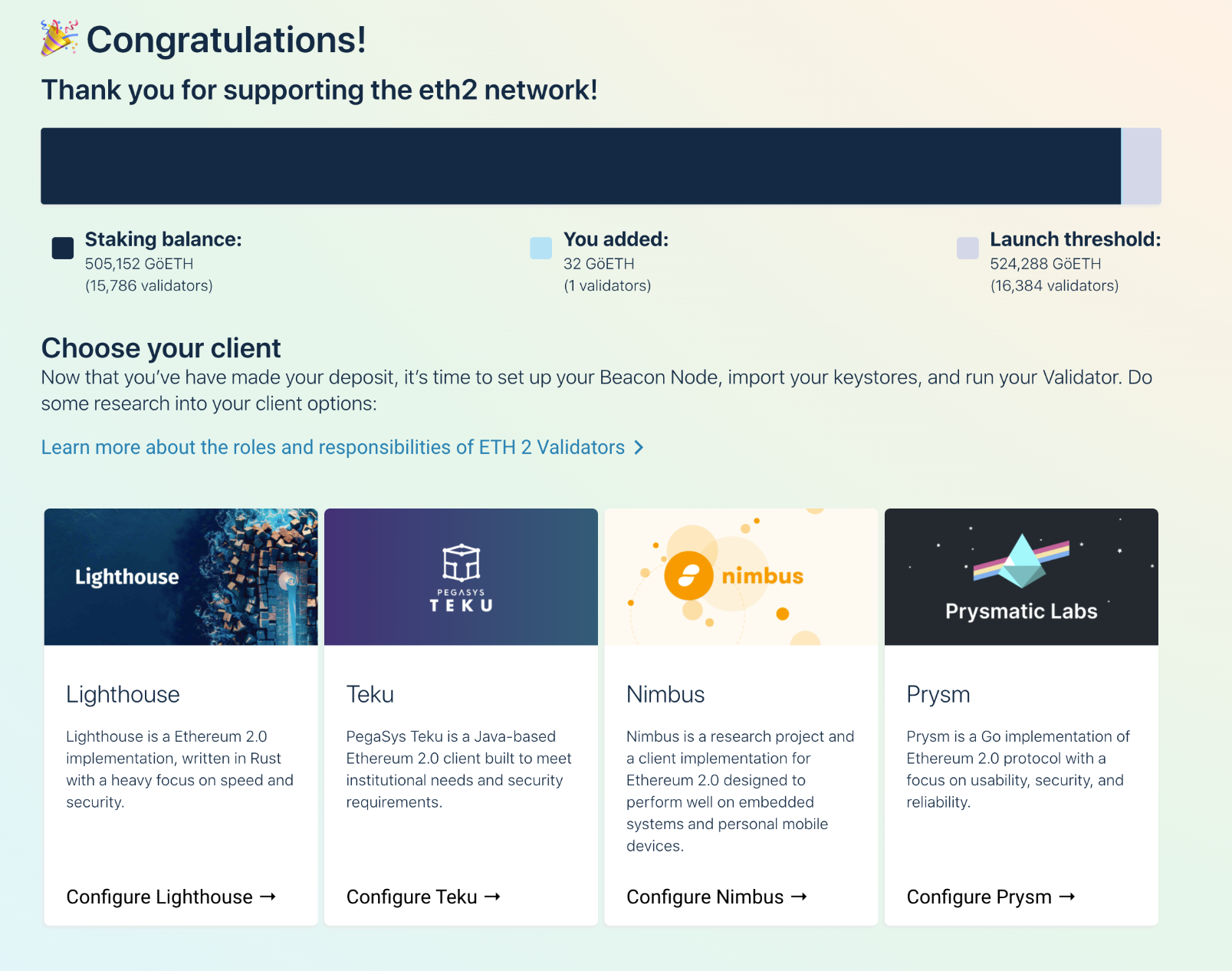
Click on initiate all transactions. This will pop open multiple instances of MetaMask, each with a 32 Göerli ETH transaction request to the Pyrmont testnet deposit contract. Confirm each transaction.

Image for post



Once all the transactions have successfully completed you are done!

Image for post



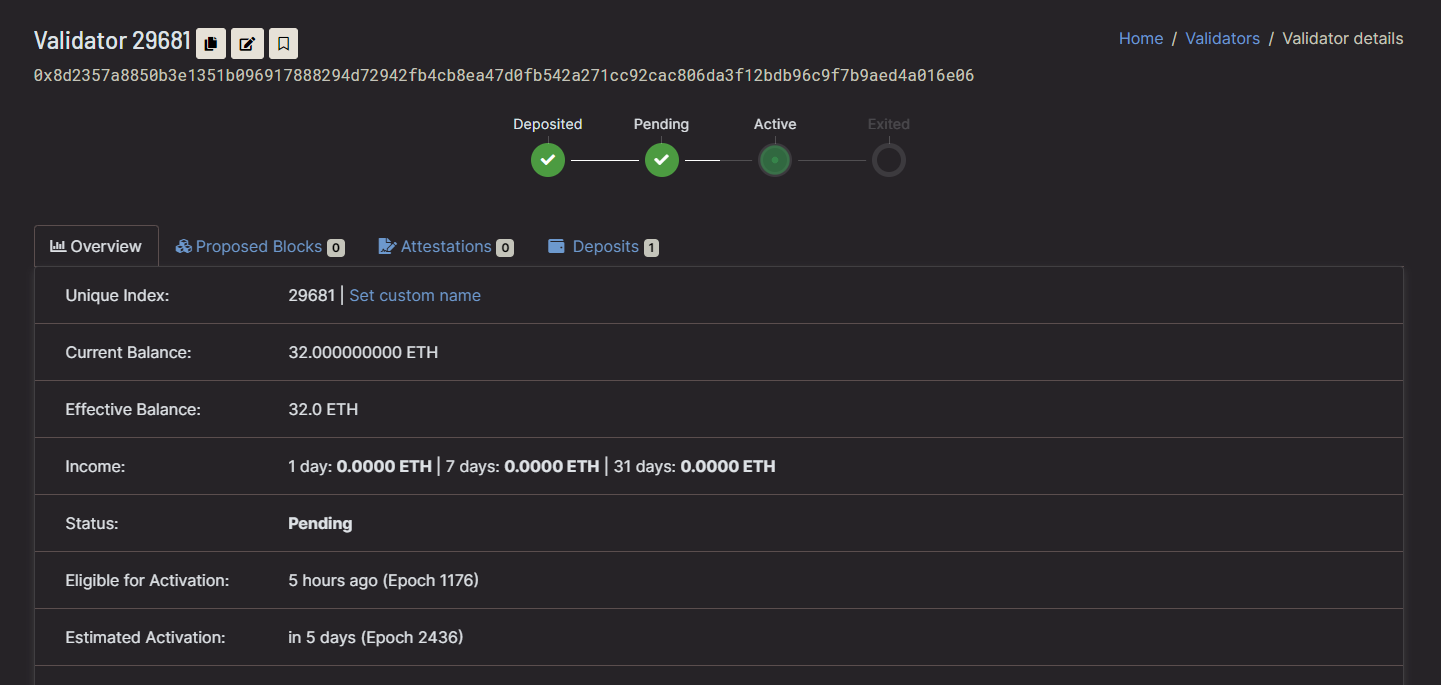
Check the Status of Your Validators

Newly added validators can take a while (hours to days) to activate. You can check the status of your keys with these steps:

1. Copy your Göerli Test Network wallet address
2. Go here: <https://pyrmont.beaconcha.in/>
3. Search your wallet address. Your keys will be shown.

Click on a key to see the**Estimated Activation**information.

Image for post



That’s it! Now let’s create the validator wallet.

Step 8 — Create the Validator Wallet

First create a directory to store the validator wallet and give the current user permissions to access it. Change <yourusername> to your logged in username. This is a temporary assignment while we create the wallet. We will reassign permissions to the validator service later.

$ sudo mkdir -p /var/lib/prysm/validator  
$ sudo chown -R <yourusername>:<yourusername> /var/lib/prysm/validator

Next we will use the Prysm validator binary we complied earlier to create a key wallet using the keys we generated in the previous step. Replace <PathToValidatorKeys> with the path to your generated validator keys. E.g. -- keys-dir=$HOME/eth2.0-deposit-cli/validator\_keys

$ cd prysm  
$ bazel run //validator:validator -- accounts import --keys-dir=<PathToValidatorKeys> --accept-terms-of-use --pyrmont

*NOTE: This command may take a while to compile the first time you run it.*

You will be asked to provide a wallet directory. This is where your new wallet will be created. Use /var/lib/prysm/validator. That is the location we have given <yourusername> access to at the beginning of this step.

Enter a wallet directory (default: /home/ethstaker/.eth2validators/prysm-wallet-v2):  
/var/lib/prysm/validator

You will be asked to provide a new wallet password. Make sure you **keep it safe**! We will need this later when configuring the validator.

Password requirements: at least 8 characters including at least 1 alphabetical character, 1 number, and 1 unicode special character. Must not be a common password nor easy to guess  
New wallet password:  
Confirm password:  
[2020-11-18 21:47:26] INFO accounts: Successfully created new wallet wallet-path=/var/lib/prysm/validator

Next you will need to enter the password you used to create the validator keys on the [Eth2 Launch Pad](https://pyrmont.launchpad.ethereum.org/) site. If you enter it correctly the accounts will be imported into the new wallet.

Enter the password for your imported accounts:  
Importing accounts, this may take a while...  
Importing accounts... 100% [==========================================] [2s:0s]  
Successfully imported 2 accounts, view all of them by running accounts list

Confirm the validator accounts have been created.

$ bazel run //validator:validator -- accounts list --pyrmont --wallet-dir /var/lib/prysm/validator --accept-terms-of-use

Shows:

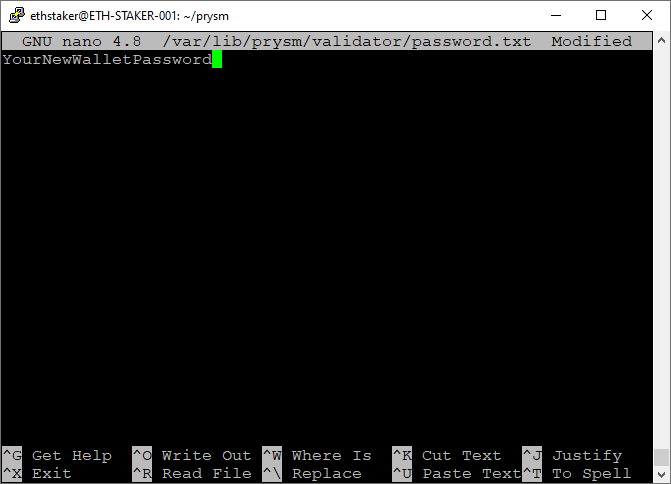
Showing 2 validator accounts  
View the eth1 deposit transaction data for your accounts by running `validator accounts list --show-deposit-dataAccount 0 | loosely-apparent-feline  
[validating public key] 0x8c8b19c544d79bdaf60d7dcc86ebaeeed5d804d2ecb4c66e5b27e19a664a81457a1c02a873a110e1d332abce5800cf7fAccount 1 | remarkably-tight-herring  
[validating public key] 0xa79583298ecbd5564fce6ccefe2e69967705aff950235dc59ae303fa210b029b565c08303eb18cf02ecc26c429059d7f

Create a file to store the wallet password so the validator can access the wallet without having to manually supply the password. The file will be named password.txt.

$ touch /var/lib/prysm/validator/password.txt  
$ sudo nano /var/lib/prysm/validator/password.txt

Add your new wallet password to the file. Save and exit.

Image for post



Protect the document by removing access for g(roup)o(ther).

$ sudo chmod go-rw /var/lib/prysm/validator/password.txt

That’s it! Now that the **validator walle**t and **password file** are configured we will set up the validator itself to run as a service.

Step 9— Configure the Validator

Setup Accounts and Directories

We will run the validator as a service so if the system restarts the process will automatically start back up again.

Create an account for the service to run under. This type of account can’t log into the server.

$ sudo useradd --no-create-home --shell /bin/false prysm-validator

We created the data directory for the validator in the previous step: /var/lib/prysm/validator. Now set directory permissions so the prysm-validator account can modify the validator account data directory.

$ sudo chown -R prysm-validator:prysm-validator /var/lib/prysm/validator

Next, copy the validator binary that we compiled earlier to the /usr/local/bin directory.

*NOTE: You will need to do this step each time you pull/build a new version of the validator binary. See****Appendix — Updating Prysm****at the end of this guide.*

$ cd ~  
$ sudo cp prysm/bazel-bin/validator/validator\_/validator /usr/local/bin

Create and Configure the Service

Create a systemd service file to store the service config.

$ sudo nano /etc/systemd/system/prysm-validator.service

Paste the following into the file exactly as it appears below with the following exceptions:

Replace <POAPstring> with your preferred text. E.g. --graffiti "abcdefg12345"

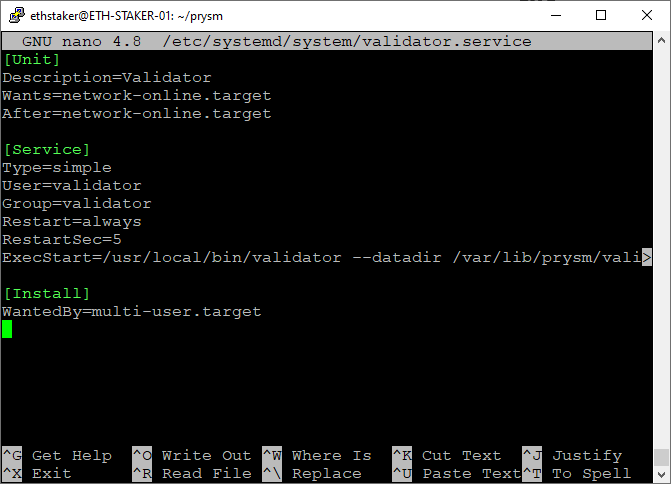
[Unit]  
Description=Validator  
Wants=network-online.target  
After=network-online.target[Service]  
Type=simple  
User=prysm-validator  
Group=prysm-validator  
Restart=always  
RestartSec=5  
ExecStart=/usr/local/bin/validator --pyrmont --datadir /var/lib/prysm/validator --wallet-dir /var/lib/prysm/validator --wallet-password-file /var/lib/prysm/validator/password.txt --graffiti "<POAPString>" --accept-terms-of-use[Install]  
WantedBy=multi-user.target

The --pyrmont flag is required to indicate we are running against the testnet.

The --accept-terms-of-use flag is required in order to be able to run the binary as a service. Using this flag indicates acceptance of the Prysm [terms of use](https://github.com/prysmaticlabs/prysm/blob/master/TERMS_OF_SERVICE.md).

Check the screen shot below for reference. Exit and save.

Image for post



Reload systemd to reflect the changes.

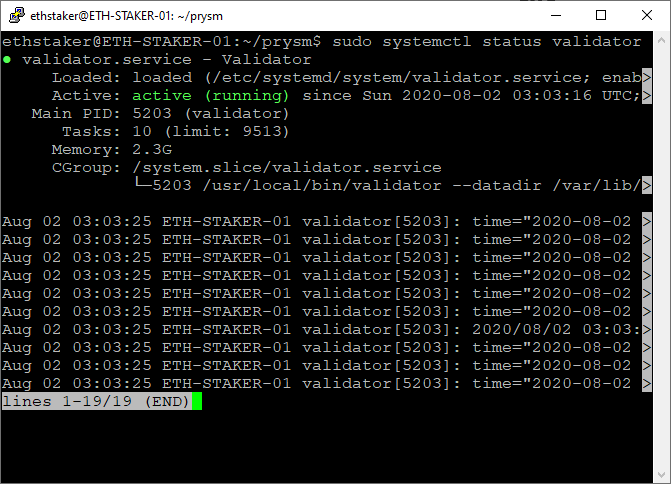
$ sudo systemctl daemon-reload

Start the service and check to make sure it’s running correctly.

$ sudo systemctl start prysm-validator  
$ sudo systemctl status prysm-validator

You should see output that looks something like this.

Image for post



If you did everything right, it should say active (running) in green text. If not then go back and repeat the steps to fix the problem. Press Q to quit.

Enable the validator service to automatically start on reboot.

$ sudo systemctl enable prysm-validator

You can check the progress by running the journal command. Press Ctrl+C to quit.

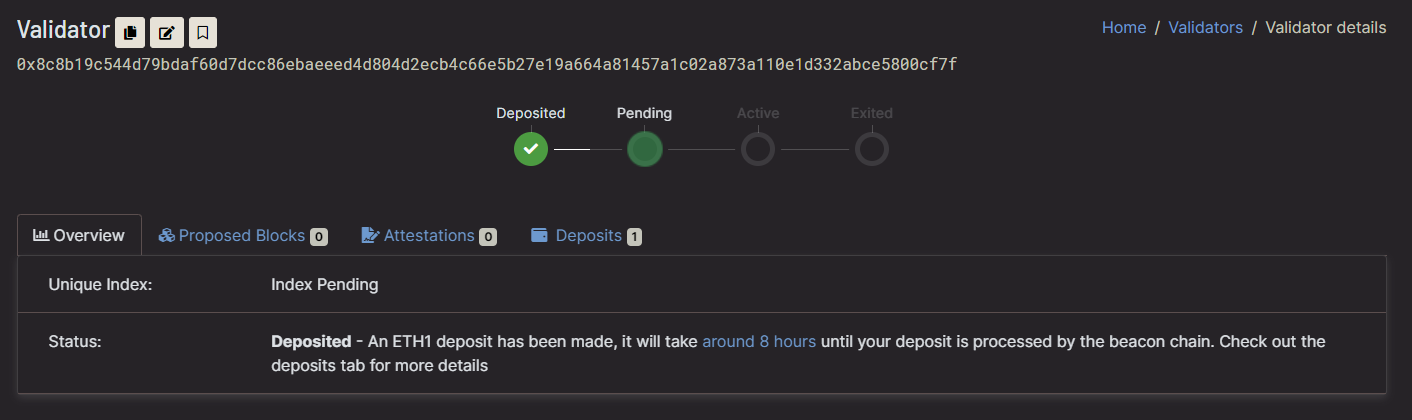
$ sudo journalctl -fu prysm-validator.service

It can take hours or days for the beacon chain to sync with your Eth1 node. It can take hours or days to activate the validation accounts(s) once the beacon chain has actually synced. The output from the validator process indicates the status.

Nov 18 21:54:35 ETH-STAKER-001 validator[119395]: time="2020-11-18 21:54:35" level=info msg="Waiting for deposit to be observed by beacon node" prefix=validator pubKey=0x8c8b19c544d7 status="UNKNOWN\_STATUS"

You can check the status of your validator(s) via [beaconcha.in](https://pyrmont.beaconcha.in/). Simply do a search for your validator public key(s) or search using your MetaMask wallet address. It may be a while before they appear on the site.

Image for post



That’s it. We have a functioning beacon-chain and validator. Congratulations: you are awesome!

Step 10 — Install Prometheus

Prometheus is an open-source systems monitoring and alerting toolkit. It runs as a service on your Ubuntu server and its job is to capture metrics. More information [here](https://prometheus.io/docs/introduction/overview/).

We are going to use Prometheus to expose runtime data from the beacon-chain and validator as well as instance specific metrics.

Create User Accounts

Accounts for the services to run under. These accounts can’t log into the server.

$ sudo useradd --no-create-home --shell /bin/false prometheus  
$ sudo useradd --no-create-home --shell /bin/false node\_exporter

Create Directories

Program and data directories.

$ sudo mkdir /etc/prometheus  
$ sudo mkdir /var/lib/prometheus

Set directory ownership. The prometheus account will manage these.

$ sudo chown -R prometheus:prometheus /etc/prometheus  
$ sudo chown -R prometheus:prometheus /var/lib/prometheus

Download Prometheus software

Adjust the version number to the latest version from the [Prometheus download page](https://prometheus.io/download/). Rpi users be sure to get the ARM binary.

$ cd ~  
$ curl -LO <https://github.com/prometheus/prometheus/releases/download/v2.22.0/prometheus-2.22.0.linux-amd64.tar.gz>

Unpack the archive. It contains two binaries and some content files.

$ tar xvf prometheus-2.22.0.linux-amd64.tar.gz

Copy the binaries to the following locations.

$ sudo cp prometheus-2.22.0.linux-amd64/prometheus /usr/local/bin/  
$ sudo cp prometheus-2.22.0.linux-amd64/promtool /usr/local/bin/

Set directory ownership. The prometheus account will manage these.

$ sudo chown -R prometheus:prometheus /usr/local/bin/prometheus  
$ sudo chown -R prometheus:prometheus /usr/local/bin/promtool

Copy the content files to the following locations.

$ sudo cp -r prometheus-2.22.0.linux-amd64/consoles /etc/prometheus  
$ sudo cp -r prometheus-2.22.0.linux-amd64/console\_libraries /etc/prometheus

Set directory and file (-R) ownership. The prometheus account will manage these.

$ sudo chown -R prometheus:prometheus /etc/prometheus/consoles  
$ sudo chown -R prometheus:prometheus /etc/prometheus/console\_libraries

Remove the downloaded archive.

$ rm -rf prometheus-2.22.0.linux-amd64.tar.gz prometheus-2.22.0.linux-amd64

Edit the Configuration File

Prometheus uses a configuration file so it knows where to scrape the data from. We will set this up here.

Open the YAML config file for editing.

$ sudo nano /etc/prometheus/prometheus.yml

Paste the following into the file taking care not to make any additional edits and exit and save the file.

global:  
 scrape\_interval: 15s # Set the scrape interval to every 15 seconds. Default is every 1 minute.  
 evaluation\_interval: 15s # Evaluate rules every 15 seconds. The default is every 1 minute.  
 # scrape\_timeout is set to the global default (10s).  
# Alertmanager configuration  
alerting:  
 alertmanagers:  
 - static\_configs:  
 - targets:  
 # - alertmanager:9093  
# Load rules once and periodically evaluate them according to the global 'evaluation\_interval'.  
rule\_files:  
 # - "first\_rules.yml"  
 # - "second\_rules.yml"  
# A scrape configuration containing exactly one endpoint to scrape:  
# Here it's Prometheus itself.  
scrape\_configs:  
 - job\_name: 'validator'  
 static\_configs:  
 - targets: ['localhost:8081']  
 - job\_name: 'beacon node'  
 static\_configs:  
 - targets: ['localhost:8080']  
 - job\_name: 'node\_exporter'  
 static\_configs:  
 - targets: ['localhost:9100']

The scrape\_configs define the output target for the different job names. We have 3 job names: validator, beacon node, and node\_exporter. The first two are obvious, the last one is for metrics related to the server instance itself (memory, CPU, disk, network etc.). We will install and configure node\_exporter below.

Set ownership for the config file. The prometheus account will own this.

$ sudo chown -R prometheus:prometheus /etc/prometheus/prometheus.yml

Finally, let’s test the service is running correctly.

$ sudo -u prometheus /usr/local/bin/prometheus \  
 --config.file /etc/prometheus/prometheus.yml \  
 --storage.tsdb.path /var/lib/prometheus/ \  
 --web.console.templates=/etc/prometheus/consoles \  
 --web.console.libraries=/etc/prometheus/console\_libraries

Output should look something like this. Press Ctrl + C to exit.

level=info ts=2020-10-21T07:18:00.434Z caller=main.go:684 msg="Server is ready to receive web requests."

Set Prometheus to Auto-Start as a Service

Create a systemd service file to store the service config which tells systemd to run Prometheus as the prometheus user, with the configuration file located in the /etc/prometheus/prometheus.yml directory, and to store its data in the /var/lib/prometheus directory.

$ sudo nano /etc/systemd/system/prometheus.service

Paste the following into the file. Exit and save.

[Unit]  
Description=Prometheus  
Wants=network-online.target  
After=network-online.target[Service]  
Type=simple  
User=prometheus  
Group=prometheus  
Restart=always  
RestartSec=5  
ExecStart=/usr/local/bin/prometheus \  
 --config.file /etc/prometheus/prometheus.yml \  
 --storage.tsdb.path /var/lib/prometheus/ \  
 --web.console.templates=/etc/prometheus/consoles \  
 --web.console.libraries=/etc/prometheus/console\_libraries[Install]  
WantedBy=multi-user.target

Reload systemd to reflect the changes.

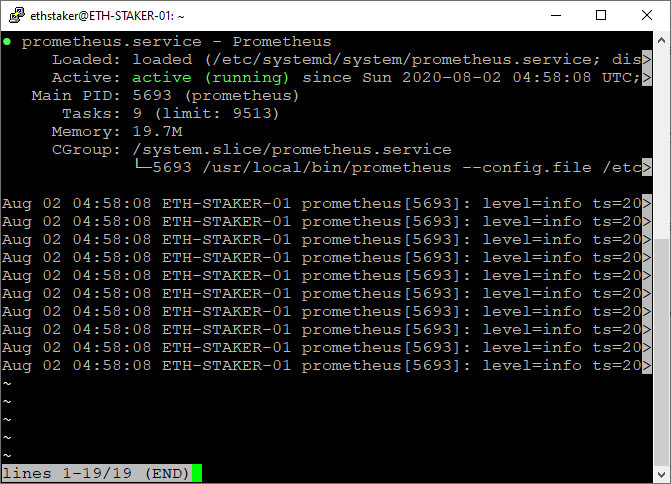
$ sudo systemctl daemon-reload

And then start the service with the following command and check the status to make sure it’s running correctly.

$ sudo systemctl start prometheus  
$ sudo systemctl status prometheus

Output should look something like this.

Image for post



If you did everything right, it should say active (running) in green. If not then go back and repeat the steps to fix the problem. Press Q to quit.

Lastly, enable Prometheus to start on boot.

$ sudo systemctl enable prometheus

Step 11 — Install Node Exporter

Prometheus will provide metrics about the beacon chain and validators. If we want metrics about our Ubuntu instance, we’ll need an extension called [Node\_Exporter](https://github.com/prometheus/node_exporter). You can find the latest stable version [here](https://prometheus.io/download/) if you want to specify a different version below. Rpi users remember to get the ARM binary.

$ cd ~  
$ curl -LO <https://github.com/prometheus/node_exporter/releases/download/v1.0.1/node_exporter-1.0.1.linux-amd64.tar.gz>

Unpack the downloaded software.

$ tar xvf node\_exporter-1.0.1.linux-amd64.tar.gz

Copy the binary to the /usr/local/bin directory and set the user and group ownership to the node\_exporter user we created above.

$ sudo cp node\_exporter-1.0.1.linux-amd64/node\_exporter /usr/local/bin  
$ sudo chown -R node\_exporter:node\_exporter /usr/local/bin/node\_exporter

Remove the downloaded archive.

$ rm -rf node\_exporter-1.0.1.linux-amd64.tar.gz node\_exporter-1.0.1.linux-amd64

Set Node Exporter to Auto-Start as a Service

Create a systemd service file to store the service config which tells systemd to run Node\_Exporter as the node\_exporter user.

$ sudo nano /etc/systemd/system/node\_exporter.service

Paste the following into the file. Exit and save.

[Unit]  
Description=Node Exporter  
Wants=network-online.target  
After=network-online.target[Service]  
User=node\_exporter  
Group=node\_exporter  
Type=simple  
ExecStart=/usr/local/bin/node\_exporter[Install]  
WantedBy=multi-user.target

Reload systemd to reflect the changes.

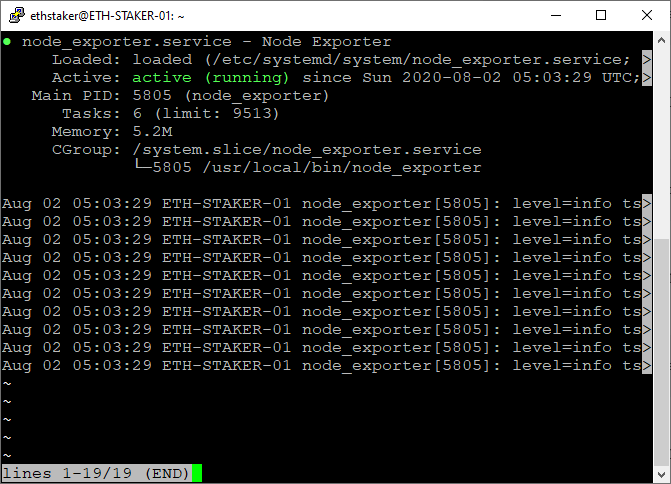
$ sudo systemctl daemon-reload

And then start the service with the following command and check the status to make sure it’s running correctly.

$ sudo systemctl start node\_exporter  
$ sudo systemctl status node\_exporter

Output should look something like this.

Image for post



If you did everything right, it should say active (running) in green. If not then go back and repeat the steps to fix the problem. Press Q to quit.

Finally, enable Node Exporter to start on boot.

$ sudo systemctl enable node\_exporter

Test Prometheus and Node Exporter

Everything should be ready to go. You may optionally test the functionality by opening a port in the firewall (see Step 2) and browsing to http://<yourserverip>:9090. From there you can run queries to view different metrics. For example try this query to see how much memory is free in bytes:

http://<yourserverip>:9090/new/graph?g0.expr=node\_memory\_MemFree\_bytes&g0.tab=1&g0.stacked=0&g0.range\_input=1h

Or try this query to see the balance for all of your validators.

http://<yourserverip>:9090/graph?g0.range\_input=1h&g0.expr=validator\_balance&g0.tab=1

Step 12 — Install Grafana

While Prometheus is our data source, Grafana is going provide our reporting dashboard capability. Let’s install it and configure a dashboard.

We will install using an APT repository because it is easier to install and update. Grafana is available in the official Ubuntu packages repository, however the version of Grafana there may not be the latest, so we will use Grafana’s official repository.

Download the Grafana GPG key with wget, then pipe the output to apt-key. This will add the key to your APT installation’s list of trusted keys.

$ wget -q -O - https://packages.grafana.com/gpg.key | sudo apt-key add -

Add the Grafana repository to the APT sources.

$ sudo add-apt-repository "deb https://packages.grafana.com/oss/deb stable main"

Refresh the apt cache.

$ sudo apt update

Make sure Grafana is installed from the repository.

$ apt-cache policy grafana

Output should look like this.

grafana:  
 Installed: (none)  
 Candidate: 7.2.1  
 Version table:  
 7.2.1 500  
 500 <https://packages.grafana.com/oss/deb> stable/main amd64 Packages  
 7.2.0 500  
 500 <https://packages.grafana.com/oss/deb> stable/main amd64 Packages  
 7.1.5 500  
 500 <https://packages.grafana.com/oss/deb> stable/main amd64  
 ...

Verify the version at the top matches the latest version shown [here](https://grafana.com/grafana/download). Then proceed with the installation.

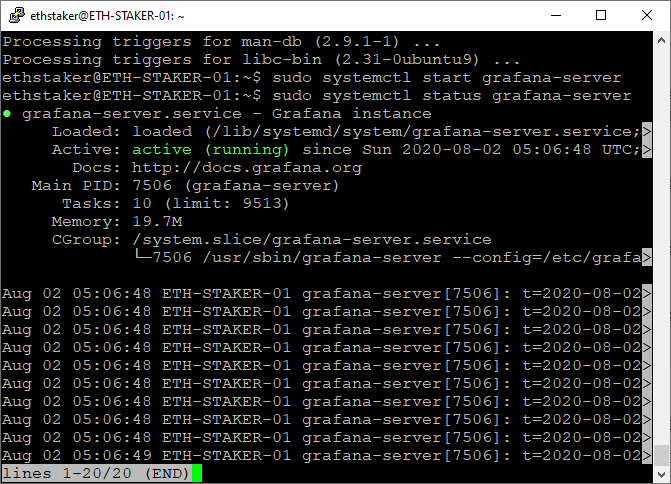
$ sudo apt install grafana

Start the Grafana server and check the status to make sure it’s running correctly.

$ sudo systemctl start grafana-server  
$ sudo systemctl status grafana-server

Output should look something like this.

Image for post



If you did everything right, it should say active (running) in green. If not then go back and repeat the steps to fix the problem. Press Q to quit.

Enable Grafana to start on boot.

$ sudo systemctl enable grafana-server

Configure Grafana Login

Great job on getting this far! Now that you have everything up and running you can go to http://<yourserverip>:3000/ in a browser and the Grafana login screen should come up.

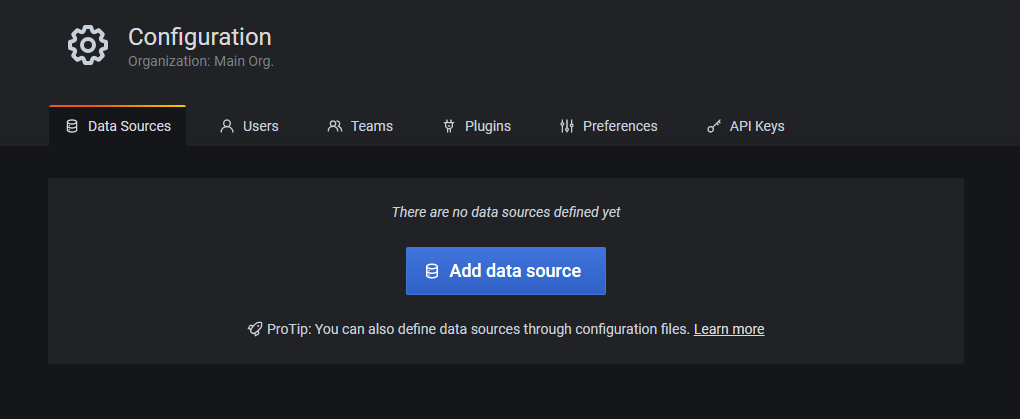
*NOTE: This is not a secure connection. Fine for testnet purposes, but for mainnet a secure HTTPS connection is required.*

Enter admin for the username and password. It will prompt you to change your password and you should definitely do that.

Configure the Grafana Data Source

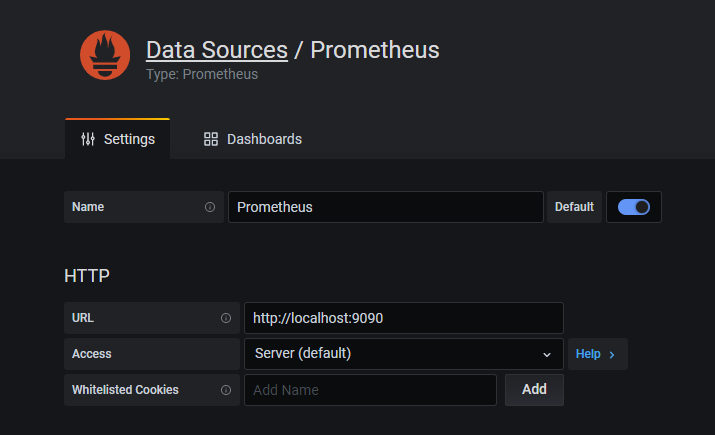
Let’s configure a datasource. Move your mouse over the gear icon on the left menu bar. A menu will pop-up — choose Data Sources.

Add data source screen.

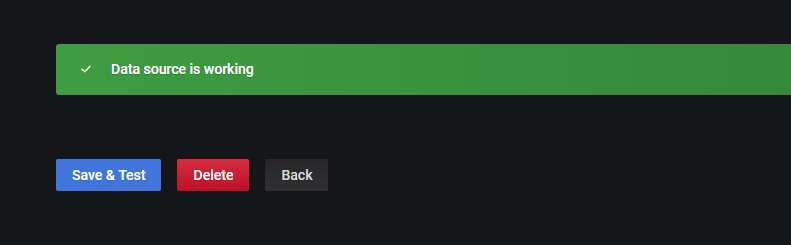


Click on Add Data Source and then choose Prometheus. Enter http://localhost:9090 for the URL then click on Save and Test.

Data source configuration.



Data source is confirmed working.

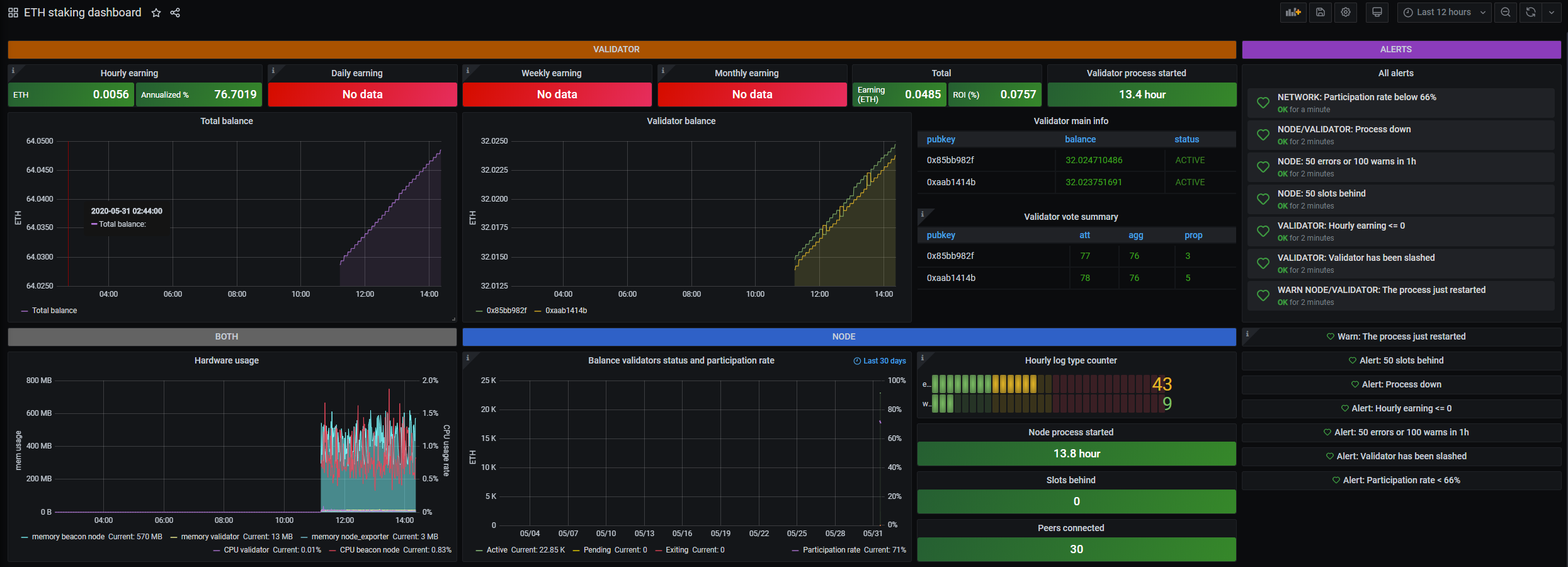


Import a Grafana Dashboard

Now let’s import a dashboard. Move your mouse over the + icon on the left menu bar. A menu will pop-up - choose Import.

Paste the JSON from [here](https://raw.githubusercontent.com/GuillaumeMiralles/prysm-grafana-dashboard/master/less_10_validators.json) (or [here](https://raw.githubusercontent.com/GuillaumeMiralles/prysm-grafana-dashboard/master/more_10_validators.json) if you have more than 10 validators) and click Load then Import. You should be able to view the dashboard. At first you may not have sufficient data, but after the testnet starts and validators are activated for a while you will see some metrics and alerts.

Ethereum staking dashboard in Grafana.



Alerts are also available through Telegram and Discord. See [here](https://docs.prylabs.network/docs/prysm-usage/monitoring/grafana-dashboard/) for instructions.

Final Remarks

Okay… That’s it! We are done! I hope you enjoyed this guide.

* A future update includes a more comprehensive dashboard (additional hardware metrics and metrics on the eth1 node).
* If you have feedback you can reach me on [Twitter](https://www.twitter.com/SomerEsat) or [Reddit](https://www.reddit.com/user/SomerEsat).
* If you liked this guide and think others would benefit from it then please share it using the [friends link](https://medium.com/@SomerEsat/4d2a86cc637b?source=friends_link&sk=4cb64bfa20247d2b5c7a50ce0a92d33b)!
* Tips appreciated: somer.eth

Appendix — Updating Prysm

If you need to update the code due to changes in the Git repository follow these steps to get the latest files and build your binaries.

Make sure you have the following pre-requisites installed to support the use of --config=release.

$ sudo apt install -y libtinfo5 # Terminal handling  
$ sudo apt-get install -y libssl-dev # OpenSSL   
$ sudo apt-get install -y libgmp-dev # GMP source to build BLS

Go [here](https://github.com/prysmaticlabs/prysm/releases) to get the release label (latest is usually best), e.g. v1.0.0-beta.3. Note that rm -r prysm will delete the prysm directory in case you have something in there you want to keep.

$ cd ~   
$ rm -r prysm  
$ git clone -b v1.0.0-beta.3 <https://github.com/prysmaticlabs/prysm.git>   
$ cd prysm  
$ bazel build //beacon-chain:beacon-chain --config=release  
$ bazel build //validator:validator --config=release

Next we stop the beacon chain and validator services and copy the binaries over to the /usr/local/bin directory and then start the services again.

$ cd ~  
$ sudo systemctl stop prysm-validator  
$ sudo systemctl stop prysm-beaconchain  
$ sudo cp prysm/bazel-bin/beacon-chain/beacon-chain\_/beacon-chain /usr/local/bin  
$ sudo cp prysm/bazel-bin/validator/validator\_/validator /usr/local/bin  
$ sudo systemctl start prysm-beaconchain  
$ sudo systemctl status prysm-beaconchain # <-- Check for errors.  
$ sudo journalctl -fu prysm-beaconchain # <-- Check for errors.  
$ sudo systemctl start prysm-validator  
$ sudo systemctl status prysm-validator # <-- Check for errors.  
$ sudo journalctl -fu prysm-validator # <-- Check for errors.

That’s it, the services have been updated.