The Ultimate Spy Mission: Setting up an Ethereum Validator Node on Google Cloud

Institution
International Academy of Espionage and Technology

Mission
Setting up and running an Ethereum validator node using Google Cloud's infrastructure.

Head Instructor
Toon aka *"Agent T"*

"In essence, the world of Ethereum validator nodes and the broader blockchain ecosystem can be viewed through the lens of espionage, with both fields valuing secrecy, resilience, and strategic collaboration."

Welcome to The International Academy of Espionage and Technology!

Dear Young Spy,

I am **Agent T**, head instructor of The International Academy of Espionage and Technology, and it is my distinct privilege to welcome you to our esteemed institution. Over the years, I've had the honor of training some of the world's most accomplished spies such as James Bond, and I see the same potential in you.

International Academy of Espionage and Technology

Our academy stands at the intersection of traditional espionage and cutting-edge technology. Here, you will embark on a transformative journey, acquiring skills that will set you apart in the realm of international intrigue:

Stealth and Infiltration
Just as a spy moves covertly, Ethereum validator nodes operate on a decentralized network, making them harder to target or shut down. This decentralization ensures that information (transactions) flows across the network seamlessly, without central points of failure.

Gadgetry
The intricate technology behind Ethereum, including its consensus algorithms and smart contract functionality, can be likened to a spy's gadgetry. Validator nodes in Ethereum use this "tech gadgetry" to validate and record transactions, ensuring the network remains active and functional.

Communication and Codes
Cryptography is at the heart of blockchain technology, ensuring secure communication between nodes and users. Ethereum transactions are encrypted, much like coded spy messages. Validator nodes "decrypt" these messages (transactions) to validate and add them to the blockchain, ensuring they remain confidential and tamper-proof.

Survival Techniques
Ethereum's decentralized structure ensures its resilience. Just as a spy might need to survive in challenging environments, Ethereum's network is designed to function and adapt even if parts of it come under attack or fail. Validator nodes help in achieving this resilience by distributing the transaction validation process across the globe.

Social Espionage
In the world of blockchain, consensus mechanisms (like Proof-of-Stake in Ethereum 2.0) require nodes to come to an agreement on the validity of transactions. This can be seen as a form of "social espionage" where validator nodes must "converse" and reach a consensus to ensure the integrity of the information (transactions) they're adding to the ledger. Additionally, the open-source community surrounding Ethereum collaborates and communicates, much like spies exchanging information, to enhance and secure the system.

While the challenges ahead are numerous, I have every confidence that you will rise to meet them.

Prepare for a thrilling journey, blending the timeless arts of espionage with the boundless possibilities of technology. We see in you a spark, a potential that can shape the future of the spy world.

Welcome to The International Academy of Espionage and Technology. Your legend begins now.

Warm regards,

Agent T
Head Instructor

The Ultimate Spy Mission

Mission Overview

Your mission, should you choose to accept it, is **to set up and run an Ethereum validator node using Google Cloud's infrastructure**, ensuring Ethereum's security and stability.

Mission Checklist

1- Understanding Blockchain Through Espionage

Imagine the world of espionage as an intricate web of operations. Just as spies carry out missions across the globe, computers (or nodes) in the blockchain network work together to record transactions.

The Mission Journal (Blockchain)
Think of the blockchain as the official mission journal of the spy world. Each time a group of spies completes a mission (or a set of transactions), their reports are bundled into a "mission folder" (block). These folders are then meticulously organized in a sequence, creating a continuous chain of missions, known as the "mission chronicle" (blockchain).

Key Features

- **Decentralization**

In our spy agency, we believe in not putting all our eggs in one basket. Instead of a single control center dictating all missions, every spy (node) has a say in the consensus process. This way, even if a base is compromised, the agency's operations remain intact.

- **Transparency**

Every mission, once completed, is broadcasted to all spies in the network. This ensures that all agents are on the same page and can verify the authenticity of each mission. It's like having a shared mission dashboard where everyone can see the latest operations.

- **Immutability**

Once a mission is added to our chronicle, it becomes a permanent record. It's as if we've written it with ink that can't be erased. This ensures that no rogue agent can tamper with past missions, guaranteeing the integrity of our records.

- **Security**

To safeguard our mission details, we use advanced cryptographic techniques. Imagine every mission folder being sealed with a unique, unbreakable seal. This ensures that even if enemies get their hands on a folder, they cannot decipher its contents.

 **Agent T** - *“In essence, young agents, the blockchain is our mission chronicle, ensuring every operation's integrity, transparency, and security in a decentralized world of espionage. Stay sharp and always trust the chronicle!”*

 **Ressources**
https://ethereum.org/en/developers/docs/intro-to-ethereum/

2- Unraveling Ethereum Through Espionage

In the vast world of international espionage, Ethereum can be likened to an advanced spy operations hub, far more than just a journal of missions. While it tracks missions like blockchain, Ethereum's true genius is its ability to program and automate spy operations through coded contracts.

The Spy Operations Center (Ethereum Platform)
Think of Ethereum as the central hub where spies can not only log their missions but also set up automatic, coded protocols called "smart contracts." These contracts trigger specific actions when certain conditions are met. Imagine setting a trap that activates only when a specific enemy agent walks into it; that's a smart contract in action!

Key Features

- **Decentralization**

Our spy network operates globally, ensuring no single point can control all missions. Every spy (node) in the Ethereum network has an equal say in verifying operations, making it robust and resistant to sabotage.

- **Transparency**

In Ethereum, every operation, every contract, and every move is broadcasted to the entire network. It's like having every spy equipped with a gadget that keeps them updated about all ongoing operations.

- **Smart Contracts**

These are like automated spy protocols. Set the conditions, and once they're met, the operations execute by themselves. Whether it's triggering a safe house alarm when an intruder is detected or releasing funds for gadgets when a mission is successful, these contracts operate without the need for manual intervention.

- **Ether (Fuel for Operations)**

Just as spies need resources to carry out their missions, Ethereum requires "Ether" to power its operations. Think of Ether as the special currency we use to fuel our gadgets and protocols, ensuring our operations run smoothly.

- **Security**

Ethereum amplifies our cryptographic defenses. Every mission, contract, and transaction is encrypted with advanced codes, making sure our secrets remain safe from enemy agents. It's like having a high-tech vault that only select agents can access.

Agent *T* - *“In summary, young operatives, Ethereum is not just a chronicle of missions. It's an advanced operations hub, empowering our spy network with automated protocols and top-tier security. As you venture deeper into the world of espionage, understanding and mastering Ethereum will be pivotal to our agency's success. Stay vigilant and trust the system!”*

Ressources

https://ethereum.org/en/developers/docs/intro-to-ethereum/

3- Ethereum Validator on Google Cloud

In the vast landscape of espionage, Google Cloud is akin to our state-of-the-art spy headquarters, equipped with the latest tech, resources, and global reach. Here, agents can set up their command centers to monitor, validate, and secure Ethereum operations.

The Global Operations Base (Google Cloud Platform)
Think of Google Cloud as our main base, complete with supercomputers, high-speed communication lines, and top-notch security. This base provides us with the infrastructure to run our Ethereum validator nodes efficiently and securely.

Key Features

- **Global Presence**

Just as our agency operates worldwide, Google Cloud's data centers span the globe, ensuring our Ethereum validator node is always online, always vigilant.

- **Top-Tier Security**

Google Cloud is fortified like the most secure spy bunkers. With advanced encryption and multiple layers of protection, our Ethereum operations remain shielded from prying eyes.

- **Scalability**

Just as a spy needs to adapt to new challenges, Google Cloud allows our Ethereum validator operations to scale up or down based on the demands of the mission. Need more computational power? Google Cloud has our back!

- **Ethereum Validator Node**

Setting up a base on Google Cloud means our validator node will be in the heart of the action, verifying Ethereum transactions and ensuring the network's integrity. This node is our listening post, ensuring every Ethereum transaction is legitimate.

- **Resource Management**

Like ensuring our agents have the gadgets and resources they need, Google Cloud offers tools to manage, monitor, and optimize our Ethereum validator node operations.

Agent *T* - *“In essence, recruits, think of Google Cloud as our ultra-secure, globally positioned spy base. When we choose to run our Ethereum validator node there, we're positioning our operations at the nexus of technology and security. Your mission, should you choose to accept it, is to harness the power of Google Cloud and ensure the security and stability of the Ethereum network. Stay sharp and trust the tech!”*

4- Role of Validator Nodes in Ethereum

Agent, just as an elite spy has specialized roles, from surveillance to infiltration, in the Ethereum universe, we have Validator Nodes. Think of them as the elite spies ensuring the Ethereum network is secure, trustworthy, and always on its toes. Their role is essential, especially as Ethereum transitions its strategy from brute force (PoW) to a more tactical and strategic approach (PoS).

Responsibilities

Validator Nodes, the top-tier spies of Ethereum, have specific assignments:

- **Block Proposal**

Just as a spy proposes a mission plan, Validator Nodes suggest new mission logs (blocks) containing vital operations (transactions).

- **Block Validation**

Before executing a mission, other elite agents (nodes) cross-check the plan, ensuring everything's up to code.

- **Consensus Participation**

Like spies agreeing on a plan, nodes reach a consensus on which mission log is legitimate.

- **Network Connectivity**

An elite spy is always in touch with the HQ. Similarly, validator nodes always maintain their connection to the Ethereum network.

Rewards

For their top-notch services, our elite spies receive commendations:

- **Block Rewards**
  For successfully proposing and verifying missions, they're rewarded with Ether, the currency of the Ethereum world.

- **Transaction Fees**
  Sometimes, they also earn bonuses for special operations within the missions.

The more resources (Ether) a spy commits, the more often they get elite assignments and rewards.

Penalties

However, elite status comes with its consequences.

- **Double Proposal**

A spy proposing two different mission plans simultaneously? Suspicious!

- **Double Vote**

Voting on two different strategies is a no-no. Stick to one plan, agent.

- **Surround Vote**

A sneaky agent trying to profit no matter which mission strategy wins? Not on our watch.

- **Unavailability**

An elite spy missing when called for a mission? That's a red flag.

- **Downtime**

Agents consistently not participating? They're risking their elite status.

Remember, spies must always be available, truthful, and follow the spy code to protect their reputation and resources.

Agent *T* - ***"****Setting up as an Ethereum Validator Node on Google Cloud is like establishing a top-tier spy base in the heart of espionage central. The stakes are high, the rewards lucrative, but the responsibility immense. Stay sharp, follow the code, and contribute to making the Ethereum world a safer place!"*

5- The Power of Google Cloud in Espionage Operations

Agent-in-training, while mastering the world of Ethereum and its Validator Nodes is vital, leveraging the right infrastructure is equally crucial. Think of Google Cloud as your high-tech underground spy lair, filled with gadgets, supercomputers, and global communication arrays. Let's uncover why Google Cloud is our partner of choice in this high-stakes world.

Ultra-Secure Operations Base

Google Cloud, with its advanced security protocols, is akin to a fortress. Using its infrastructure ensures:

- **Advanced Encryption**

Your validator operations, much like our top-secret files, remain encrypted and shielded from adversaries.

- **Regular Security Updates**

Constantly updating defense mechanisms against the ever-evolving threats in the digital realm.

Global Coverage

Being globally present ensures that:

- **Always Online**

Like a spy satellite that never sets, your Ethereum validator node is always operational, tracking and verifying.

- **Low Latency**

Speed is everything in our world. Google Cloud's global presence ensures lightning-fast responses, crucial for timely block validations.

Scalability on Demand

In the unpredictable world of espionage, flexibility is vital.

- **Dynamic Resource Allocation**

Should a mission demand more computational power or storage, Google Cloud allows you to scale up instantly.

- **Pay-as-you-go**

Efficiently manage resources. Use what you need, when you need it, and pay accordingly.

Robust Monitoring & Alerts

Stay ahead of the game with:

- **Real-time Monitoring**

Just as our radars scan for threats, Google Cloud offers tools to constantly monitor your node's health and performance.

- **Instant Alerts**

Be the first to know if there's any suspicious activity or if your node faces any issues, ensuring you can act promptly.

Comprehensive Backup Systems

In case of unforeseen events:

- **Automated Backups**

Google Cloud continuously backs up your data. Should things go south, you have a safety net.

- **Disaster Recovery**

In a worst-case scenario, like a good escape plan, you can quickly recover and restore your operations.

Agent *T* - *"Young agent, in the realm of Ethereum operations, just as in espionage, having the right tools and infrastructure is half the battle won. Google Cloud isn't just a platform; it's our strategic ally, ensuring our Ethereum validator node operations run flawlessly, securely, and efficiently. Harness its power, and you'll be one step ahead in the game. Happy spying!"*

6- Gearing Up for the Mission

Welcome, agent-in-training! To operate effectively in the field of Ethereum espionage, you need the right gadgets and tools - hardware and software. Think of running an Ethereum validator node as setting up your covert ops center. Here's your detailed mission briefing:

Hardware Essentials

Just as you wouldn't embark on a mission without the best spy gear, running an Ethereum validator node requires specific hardware:

- **Processor (CPU)**

A modern multi-core processor. Think of it as the brain of your operations.

- **Memory (RAM)**

The more memory you have, the faster you can process Ethereum's operations.

- **Storage (SSD)**

SSD (Solid State Drive) is the high-speed vault where Ethereum's vast data and transaction logs get stored. Always have a contingency plan and prepare additional storage for backups.

- **Network (Bandwidth)**

A reliable and fast connection ensures you're always in sync with Ethereum HQ.
Opt for wired over wireless for security and speed.

	Minimum	**Recommended**
CPU	CPU with 2+ cores	Fast CPU with 4+ cores
RAM	8 GB RAM	16+ GB RAM
SSD	2 TB SSD	Fast SSD with 2+ TB
Bandwidth	10+ MBit/s bandwidth	25+ MBit/s bandwidth

Software Arsenal

In the digital realm of Ethereum, your software tools are as vital as physical gadgets:

- **Operating System (OS)**

Preferred choices are Linux (Ubuntu, CentOS) or MacOS. They're like your base of operations - stable and secure.

- **Ethereum Clients**

Softwares enable your ops center to connect to the Ethereum network.

  - **Execution client**

Before becoming an Ethereum validator, you need to sync your node with the Ethereum mainnet. This involves installing the execution client and syncing with the Ethereum mainnet.

  - **Consensus client**

After syncing with Ethereum mainnet, you can set up your Ethereum validator node using the Consensus client. The consensus client handles the PoS consensus mechanism and validator responsibilities.

- **Monitoring & Alert Tools**

Software like **Grafana** or **Prometheus** to keep an eye on your node's performance. The digital equivalent of surveillance cameras.

Security Protocols

Espionage and security go hand in hand:

- **Firewall rules**

Protect your ops center from external threats.

- **Regular Software Updates**

Always keep your software arsenal up to date. Outdated tools can be exploited.

Agent *T* - "*Young agent, remember, setting up your Ethereum validator node with the right hardware and software is just the beginning. But with the perfect blend of power, speed, and security, you'll be prepared to tackle any mission in the Ethereum universe. Equip yourself well, and the world of decentralized finance is yours to command. Until the next briefing, stay covert and operate efficiently!"***

7- Consensus and Execution Clients in Blockchain

Agents, remember, the success of the mission isn't just about choosing the right gear but understanding the terrain and deploying resources wisely. Ensure your team reflects the mission's objectives and challenges.

Consensus Clients - The Analysts

Consensus Clients are like your team of intelligence analysts. They work in the background, making sure all spies have the correct intel and that they agree on it.

- **Accuracy vs. Speed**

Some consensus clients process information faster but might be more prone to errors. It's like choosing between a rookie analyst who's quick but might miss some details versus a seasoned analyst who's methodical but slower.

- **Resource Consumption**

Some consensus clients might be resource-intensive. It's like having an analyst who needs a lot of gadgets and tools to get the job done.

Execution Clients - The Field Agents

Execution Clients are your field agents. They take the intel and act on it, interacting with the external world and ensuring the mission's execution.

- **Flexibility vs. Complexity**

Some execution clients can handle a wide variety of tasks but might be harder to manage. It's like choosing a spy who's trained in multiple disciplines but requires more briefing.

- **Performance**

Some execution clients perform actions more swiftly, similar to a spy who can infiltrate a building faster than others.

- **Community Support**

Just as a spy with more allies can get more assistance during missions, an execution client with a larger community gets more updates and patches.

Case Study

Imagine you're planning an operation to infiltrate a high-tech facility with Besu (Consensus) and Lighthouse (Execution)

- **Besu**

It's like hiring an intelligence analyst (consensus client) who's well-versed with multiple intelligence networks (compatible with various Ethereum testnets). Besu is known for its flexibility as it supports both private and public network modes. While it offers versatility, it might need more resources and setup compared to some more straightforward analysts.

- **Lighthouse**

For the field operation, you've chosen a field agent (execution client) known for being efficient and lean. Lighthouse focuses on speed and performance, ensuring that your operations (transactions) are executed swiftly. While Lighthouse is optimized for speed, it might not have as many fancy features (extensions) as some other agents.

In our spy operation, we need an analyst who can work with different intel sources (Besu's compatibility) and a field agent who can get the job done quickly and efficiently (Lighthouse's performance). Together, they form a team that is versatile and efficient, ideal for a mission that requires adaptability and speed.

Agent *T* - "*Agent, remember that in the world of blockchain, as in espionage, every choice comes with trade-offs. Understanding your mission's requirements (or your blockchain's needs) is crucial to select the perfect team of tools and software."***

8- The Double-Edged Sword - Pros and Cons of Running an Ethereum Validator Node

Attention, agent! Just as a spy must weigh the benefits and risks of every mission, you must understand the advantages and pitfalls of running an Ethereum validator node. By the end of this briefing, you'll see the bigger picture and understand why tools like Google Cloud can be your secret weapon.

Advantages of Running a Validator Node

- **Decentralized Power**

Be a part of the decentralized revolution, reinforcing Ethereum's strength against central threats.

- **Earnings & Rewards**

Successful validation earns you Ether, the currency of the Ethereum realm.

- **First-hand Intelligence**

Direct access to the latest network activities and transactions, keeping you ahead in the intel game.

- **Reputation Boost**

As you consistently validate and propose blocks, you gain reputation and trust within the Ethereum network.

Pitfalls and Challenges

- **Initial Investment**

Just as field equipment isn't cheap, setting up a robust node requires a good initial investment in hardware.

- **Operational Costs**

Your covert ops center, the validator node, requires continuous power and internet, adding to operational expenses.

- **Slashing Risks**

Mistakes or misbehavior can lead to penalties, affecting both your reputation and financial reserves.

- **Technical Challenges**

Running a validator node isn't just plug-and-play; it requires ongoing technical oversight and adjustments.

- **Security Threats**

The digital realm is fraught with cyber threats, and your node is a potential target for hackers and adversaries.

Why Google Cloud is the next Destination

While the realm of Ethereum validation is laden with challenges, our next chapter will delve into how Google Cloud can be your secret ally.

- **Operational Efficiency**

How Google Cloud's infrastructure can reduce your operational hassles.

- **Security Enhancement**

Google Cloud's advanced security measures to safeguard your node.

- **Cost Management**

Harnessing the power of the cloud to manage and potentially reduce operational costs.

- **High Availability**

Ensuring your node remains operational, reducing the risk of penalties due to downtime.

 **Agent *T* - *"While running an Ethereum validator node offers many rewards, it's not without its challenges. However, with the right tools and allies, even the most daunting missions become achievable. Gear up, agent, as we dive deeper into the world of Google Cloud in our next chapter and discover how it can turn the tables in your Ethereum operations. The world of Ethereum awaits your expertise!"***

9- Deploying Ether Operations with Google Cloud

Greetings, agents-in-training! In the ever-evolving field of Ethereum espionage, leveraging cutting-edge infrastructure can give you a strategic edge. Enter Google Cloud. Let's decrypt the benefits and understand how Google Cloud can assist you in meeting the hardware and software requisites for an Ethereum validator node mission.

Elite Hardware Resources

Google Cloud offers a top-tier arsenal, designed for covert operations:

- **Computational Power**

Customizable virtual machines equipped with high-performance CPUs. These are the digital counterparts of our field's best vehicles.

- **RAM**

Generous memory options to ensure smooth operations, keeping your node agile and responsive.

- **State-of-the-art SSDs**

Fast, reliable, and scalable storage solutions. Think of these as the advanced safe houses for your crucial Ethereum data.

- **Global Network**

Lightning-speed connectivity with redundancy ensures your node remains operational 24/7, akin to having a global spy satellite network.

Software Armory

Google Cloud provides an array of software tools and platforms, optimized for clandestine operations:

- **Seamless OS Deployment**

Choose from Linux distributions or other OS options, pre-configured for instant deployment, saving valuable mission time.

- **Automated Backup and Restore**

Scheduled backups to ensure your node's data is safe. The perfect escape plan for any sticky situation.

- **Monitoring Tools**

Integrated solutions like Stackdriver let you keep a hawk's eye on your node, ensuring optimal performance and immediate anomaly detection.

Fortified Security Measures

In the spy world, security is paramount. Google Cloud is no different:

- **Advanced Firewall**

A virtual barrier that guards against any unauthorized entry. This is your digital perimeter defense.

- **Data Encryption**

Both in-transit and at-rest encryption protocols, ensuring your Ethereum operations remain confidential.

- **Identity & Access Management (IAM)**

Control who accesses what. It's like vetting agents before giving them access to top-secret files.

- **Regular Software Patching**

Google Cloud continuously updates its systems, ensuring you always have the latest security and functionality updates.

Agent T - "*Agents, in the world of Ethereum espionage, Google Cloud is akin to a state-of-the-art spy headquarters. It offers you the best in class hardware, sophisticated software tools, and fortified security. Deploying your Ethereum validator node here means you're not just in the game, but ahead of it. Equip yourself with Google Cloud, and let's redefine Ethereum intelligence!"*

10- The Final Countdown - Set and Run

Mission Briefing by Agent T

Goerli Test Network

You are being deployed to the Goerli testnet. IMPORTANT: Under NO circumstances are you to transfer mainnet ETH to Goerli testnet. The consequences are irreversible. Proceed with the utmost caution.

Also, the given mission guide is NOT to be utilized for staking operations on the Ethereum mainnet.

Intelligence Disclaimer

Bear in mind that the intel dossier provided (the guide) is for recon and knowledge-gathering only. HQ does not endorse its use as a strategic action plan. The accuracy of the intel cannot be fully vouched for, and the handler is not accountable for any potential mission compromises. A comprehensive disclaimer code can be located at the dossier's conclusion. Review thoroughly before action.

Final countdown

1. Creating a Google Cloud Project and Virtual Machine Instance
2. Preparing and securing the Server
3. Configuring Swap Space and Timekeeping
4. Setting up JWT for Client-Client Communication on Your Server
5. Setting-up an Execution Client - BEZU
6. Setting Up the Consensus Client - LIGHTHOUSE
7. Generating Staking Data
8. Importing Validator Keys
9. Configuring the Beacon Node Service
10. Configuring the Validator Service
11. From “in Progress” to “Fully Synced” Client
12. Funding the Validator
13. Monitoring your Validator

10.1- Creating a Google Cloud Project and Virtual Machine Instance

Open Your Web Browser
Begin by opening your preferred web browser on your computer.

Visit the Google Cloud Platform
In the address bar, type in "Google Cloud Console" or simply visit console.cloud.google.com.

Selecting a Project
Once the page loads, you'll see a dropdown at the top of the screen labeled "Select a Project." Click on it.

Create a New Project
In the popup window that appears, you'll see a list of any existing projects you have. To create a new one, click on the "New Project" button, usually found in the top right corner.

Name Your Project
A new page will load asking for project details. In the "Project Name" field, type in a name for your project. For example, you can name it "spy-node."

After giving your project a name, follow any additional prompts or instructions, and then click the "Create" button to finalize your new project.

Agent T *- "Now you've successfully created a new project in the Google Cloud Console!".*

Locate the Left-Hand Menu
On the main dashboard of the Google Cloud Console, you'll find a vertical menu bar on the left side of your screen. This contains various options and services offered by Google Cloud.

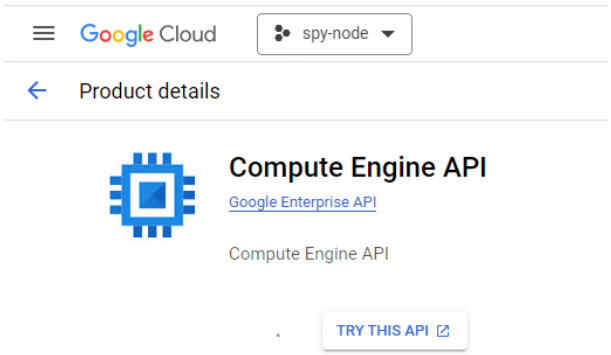
Access 'APIs & Services'
Scroll through the menu options and click on the one titled "APIs & Services." This will expand to show more specific choices.

Navigate to 'Library'
Under "APIs & Services," click on the "Library" option. This will take you to a page showcasing all the available APIs and services.

Search for the Desired API
At the top of the "Library" page, you'll see a search bar. Type "Compute Engine API" into this bar and press Enter.

Select 'Compute Engine API'
From the search results, locate and click on the "Compute Engine API" option. This will lead you to a detailed page about this specific API.

Enable the API
On the Compute Engine API page, you'll see a large "Enable" button, usually in blue. Click on this button to activate the Compute Engine API for your chosen project.



Agent T *- "By following these steps, you've successfully enabled the Compute Engine API for your project in the Google Cloud Console!".*

Navigate to 'Compute Engine'
In the left-side menu bar of the Google Cloud Console, look for and click on "Compute Engine."

Go to 'VM instances'
Under the "Compute Engine" option, select "VM instances." You'll be directed to a page displaying all your virtual machine instances, if any.

Start Creating a New Instance
On the "VM instances" page, you'll find a "Create Instance" button, usually at the top. Click on it.

Naming Your Instance
A form will appear, prompting you to fill out details for your new VM. In the "Name" field, type your desired instance name, like "james-bond-node."

Selecting Region and Zone
Scroll down to find the "Region" and "Zone" dropdown menus. Pick a region and zone that suits your preferences or requirements.

Choosing Machine Configuration

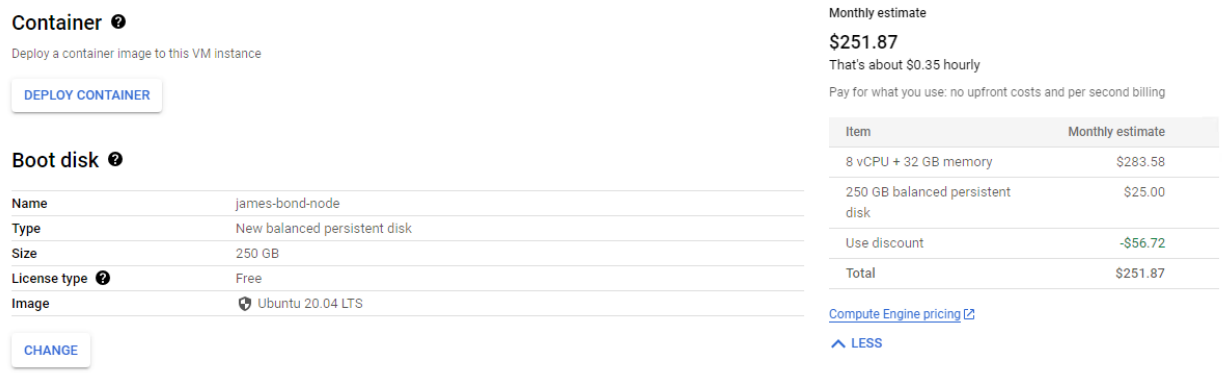
Further down, you'll see options for "Machine type." For running an Ethereum validator node, selecting "n2-standard-8" from the dropdown is a good choice.

Configuring the Boot Disk
Below the machine configuration, there's a section labeled "Boot disk." Click on "Change" to open a new window.
In this window, pick "Ubuntu" as the operating system from the list.
For the disk size, select "2000 GB" (Note: In the guide, 250 GB is selected due to free trial limitations, but it is way better using a larger size here).

Adjusting Firewall Settings
As you continue scrolling, you'll come across the "Firewall" section. Here, ensure you tick both checkboxes labeled "Allow HTTP traffic" and "Allow HTTPS traffic."

Finalizing the Creation
Once all settings are to your liking, locate and click on the "Create" button.

Agent T *- "Your new VM instance with the specified configurations will now be set up and running in a few moments!".*



VM instances

Filter Enter property name or value

Status	Name	Zone	Recommendations	In use by	Internal IP	External IP	Connect
✓	james-bond-node	us-central1-a			10.128.0.5 (nic0)	34.121.181.178 (nic0)	SSH

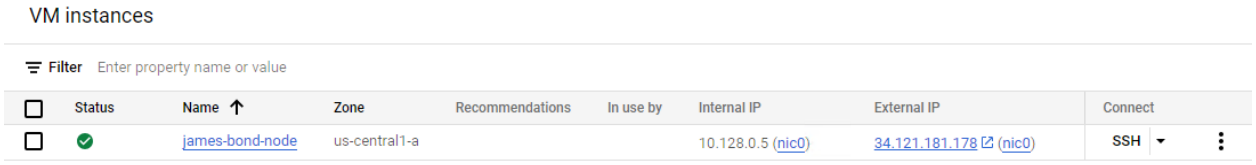
Agent T *- "Google cloud platform has just relayed classified intel for our most esteemed operatives. We've secured a financial grant for your next mission: a $300 free trial allocation!".*

10.2- Preparing and securing the Server

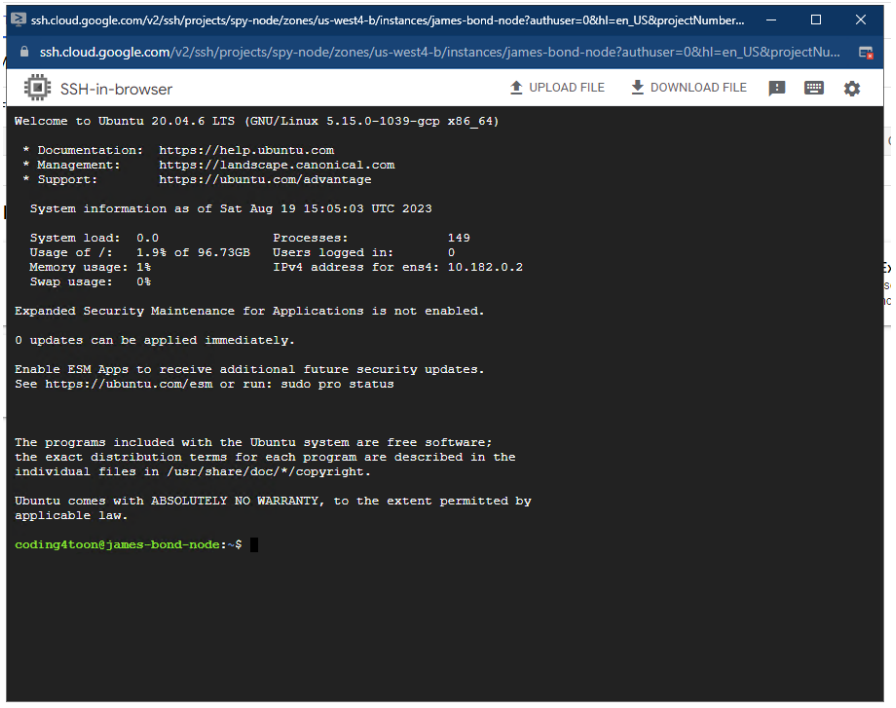
Access Your Virtual Machines
In the left-side menu bar of the Google Cloud Console, look for and click on "Compute Engine." Then, choose "VM instances" to see a list of all your virtual machine instances.

Locate Your VM
Among the listed VM instances, find the one you want to connect to. In our previous examples, it was named "james-bond-node."

Connect Using SSH
Next to your VM's name, you'll find several icons. One of them is the SSH (Secure Shell) button. Simply click on the SSH button. Google Cloud Platform makes it easy by automatically launching a new window or tab in your browser with a terminal connected directly to your VM.



After a few moments, you'll be connected to your server, and you can start issuing commands right from the browser window.



Agent T *- "Thanks to Google Cloud Platform's intuitive design, connecting to your server via SSH is a breeze, even for those new to the world of cloud computing!".*

Update Your System Software

Before you do anything, you should make sure that your system's software is updated. This includes all the packages and libraries installed.

$ sudo apt -y update && sudo apt -y upgrade

```
Get:32 http://security.ubuntu.com/ubuntu focal-security/multiverse Translation-en [5504 B]
Get:33 http://security.ubuntu.com/ubuntu focal-security/multiverse amd64 c-n-f Metadata [548 B]
Fetched 20.0 MB in 3s (6019 kB/s)
Reading package lists... Done
Building dependency tree
Reading state information... Done
All packages are up to date.
Reading package lists... Done
Building dependency tree
Reading state information... Done
Calculating upgrade... Done
The following packages were automatically installed and are no longer required:
  libatasmart4 libblockdev-fs2 libblockdev-loop2 libblockdev-part-err2 libblockdev-part2 libblockdev-swap2
  libblockdev-utils2 libblockdev2 libmbim-glib4 libmbim-proxy libmm-glib0 libnspr4 libnss3 libnuma1
  libparted-fs-resize0 libqmi-glib5 libqmi-proxy libudisks2-0 libxmlb2 usb-modeswitch usb-modeswitch-data
Use 'sudo apt autoremove' to remove them.
#
# You can verify the status of security fixes using the `pro fix` command.
# E.g., a recent Ruby vulnerability can be checked with: `pro fix USN-6219-1`
# For more detail see: https://ubuntu.com/security/notices/USN-6219-1
#
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
coding4toon@james-bond-node:~$
```

Perform a Distribution Upgrade

Next, you'll want to make sure the core parts of your Ubuntu system are updated to the latest version. Then, remove any unnecessary packages.

$ sudo apt dist-upgrade && sudo apt autoremove

```
Removing libblockdev-part2:amd64 (2.23-2ubuntu3) ...
Removing libblockdev-part-err2:amd64 (2.23-2ubuntu3) ...
Removing libblockdev-swap2:amd64 (2.23-2ubuntu3) ...
Removing libblockdev2:amd64 (2.23-2ubuntu3) ...
Removing libblockdev-utils2:amd64 (2.23-2ubuntu3) ...
Removing libqmi-proxy (1.30.4-1~ubuntu20.04.1) ...
Removing libqmi-glib5:amd64 (1.30.4-1~ubuntu20.04.1) ...
Removing libmbim-proxy (1.26.2-1~ubuntu20.04.1) ...
Removing libmbim-glib4:amd64 (1.26.2-1~ubuntu20.04.1) ...
Removing libmm-glib0:amd64 (1.18.6-1~ubuntu20.04.1) ...
Removing libnss3:amd64 (2:3.49.1-1ubuntu1.9) ...
Removing libnspr4:amd64 (2:4.25-1) ...
Removing libnuma1:amd64 (2.0.12-1) ...
Removing libparted-fs-resize0:amd64 (3.3-4ubuntu0.20.04.1) ...
Removing libudisks2-0:amd64 (2.8.4-1ubuntu2) ...
Removing libxmlb2:amd64 (0.3.6-2build1~20.04.1) ...
Removing usb-modeswitch (2.5.2+repack0-2ubuntu3) ...
Removing usb-modeswitch-data (20191128-3) ...
Processing triggers for man-db (2.9.1-1) ...
Processing triggers for libc-bin (2.31-0ubuntu9.9) ...
coding4toon@james-bond-node:~$
```

Reboot Your VM

It's a good practice to reboot your system after major upgrades to ensure everything loads correctly.

$ sudo reboot

Install the Latest Version of Java

Let's make sure you have the latest version of Java Development Kit (JDK) installed. First, update your software list again.

$ sudo apt update

```
coding4toon@james-bond-node:~$ sudo apt update
Hit:1 http://us-central1.gce.archive.ubuntu.com/ubuntu focal InRelease
Hit:2 http://us-central1.gce.archive.ubuntu.com/ubuntu focal-updates InRelease
Hit:3 http://us-central1.gce.archive.ubuntu.com/ubuntu focal-backports InRelease
Hit:4 http://security.ubuntu.com/ubuntu focal-security InRelease
Reading package lists... Done
Building dependency tree
Reading state information... Done
All packages are up to date.
coding4toon@james-bond-node:~$
```

Now, install the latest OpenJDK version:

$ sudo apt install openjdk-17-jdk

```
update-alternatives: using /usr/lib/jvm/java-17-openjdk-amd64/bin/jrunscript to provide /usr/bin/jrunscript (jr
unscript) in auto mode
update-alternatives: using /usr/lib/jvm/java-17-openjdk-amd64/bin/jshell to provide /usr/bin/jshell (jshell) in
 auto mode
update-alternatives: using /usr/lib/jvm/java-17-openjdk-amd64/bin/jstack to provide /usr/bin/jstack (jstack) in
 auto mode
update-alternatives: using /usr/lib/jvm/java-17-openjdk-amd64/bin/jstat to provide /usr/bin/jstat (jstat) in au
to mode
update-alternatives: using /usr/lib/jvm/java-17-openjdk-amd64/bin/jstatd to provide /usr/bin/jstatd (jstatd) in
 auto mode
update-alternatives: using /usr/lib/jvm/java-17-openjdk-amd64/bin/serialver to provide /usr/bin/serialver (seri
alver) in auto mode
update-alternatives: using /usr/lib/jvm/java-17-openjdk-amd64/bin/jhsdb to provide /usr/bin/jhsdb (jhsdb) in au
to mode
Setting up libgtk2.0-0:amd64 (2.24.32-4ubuntu4) ...
Setting up humanity-icon-theme (0.6.15) ...
Setting up libgail18:amd64 (2.24.32-4ubuntu4) ...
Setting up libgtk2.0-bin (2.24.32-4ubuntu4) ...
Setting up libgail-common:amd64 (2.24.32-4ubuntu4) ...
Setting up openjdk-17-jre:amd64 (17.0.8+7-1~20.04.2) ...
Setting up ubuntu-mono (19.04-0ubuntu3) ...
Setting up openjdk-17-jdk:amd64 (17.0.8+7-1~20.04.2) ...
update-alternatives: using /usr/lib/jvm/java-17-openjdk-amd64/bin/jconsole to provide /usr/bin/jconsole (jconso
le) in auto mode
Processing triggers for ca-certificates (20230311ubuntu0.20.04.1) ...
Updating certificates in /etc/ssl/certs...
0 added, 0 removed; done.
Running hooks in /etc/ca-certificates/update.d...

done.
done.
Processing triggers for mime-support (3.64ubuntu1) ...
Processing triggers for libc-bin (2.31-0ubuntu9.9) ...
Processing triggers for systemd (245.4-4ubuntu3.22) ...
Processing triggers for man-db (2.9.1-1) ...
Processing triggers for libgdk-pixbuf2.0-0:amd64 (2.40.0+dfsg-3ubuntu0.4) ...
coding4toon@james-bond-node:~$
```

Install Snappy Java Library

Snappy Java is a Java library for the Snappy compressor/decompressor. To install it, use:

$ sudo apt-get install libsnappy-java

```
coding4toon@james-bond-node:~$ sudo apt-get install libsnappy-java
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libsnappy-jni libsnappy1v5
The following NEW packages will be installed:
  libsnappy-java libsnappy-jni libsnappy1v5
0 upgraded, 3 newly installed, 0 to remove and 0 not upgraded.
Need to get 95.7 kB of archives.
After this operation, 196 kB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://us-central1.gce.archive.ubuntu.com/ubuntu focal/main amd64 libsnappy1v5 amd64 1.1.8-1build1 [16.7
kB]
Get:2 http://us-central1.gce.archive.ubuntu.com/ubuntu focal/universe amd64 libsnappy-jni amd64 1.1.7.3-1build1
 [6760 B]
Get:3 http://us-central1.gce.archive.ubuntu.com/ubuntu focal/universe amd64 libsnappy-java all 1.1.7.3-1build1
[72.3 kB]
Fetched 95.7 kB in 0s (443 kB/s)
Selecting previously unselected package libsnappy1v5:amd64.
(Reading database ... 77771 files and directories currently installed.)
Preparing to unpack .../libsnappy1v5_1.1.8-1build1_amd64.deb ...
Unpacking libsnappy1v5:amd64 (1.1.8-1build1) ...
Selecting previously unselected package libsnappy-jni.
Preparing to unpack .../libsnappy-jni_1.1.7.3-1build1_amd64.deb ...
Unpacking libsnappy-jni (1.1.7.3-1build1) ...
Selecting previously unselected package libsnappy-java.
Preparing to unpack .../libsnappy-java_1.1.7.3-1build1_all.deb ...
Unpacking libsnappy-java (1.1.7.3-1build1) ...
Setting up libsnappy1v5:amd64 (1.1.8-1build1) ...
Setting up libsnappy-jni (1.1.7.3-1build1) ...
Setting up libsnappy-java (1.1.7.3-1build1) ...
Processing triggers for libc-bin (2.31-0ubuntu9.9) ...
coding4toon@james-bond-node:~$
```

 ***Agent T*** *- "By following these steps, you'll ensure that your system is up-to-date and ready for any task!".*

Enhancing Security with SSH Port Change

Choosing a New SSH Port

Choose a port between 1024 and 49151. This range is for registered and dynamic/private ports. We'll use 7007 as an example.

 ***Agent T*** *- "Why Change the SSH Port? The default SSH port is 22. Because it's the default, attackers often target this port in brute-force attacks. By changing the port, you reduce your exposure to these automated attacks."*

Check if the Port is in Use

Before changing the SSH port, make sure the port you chose isn't in use.

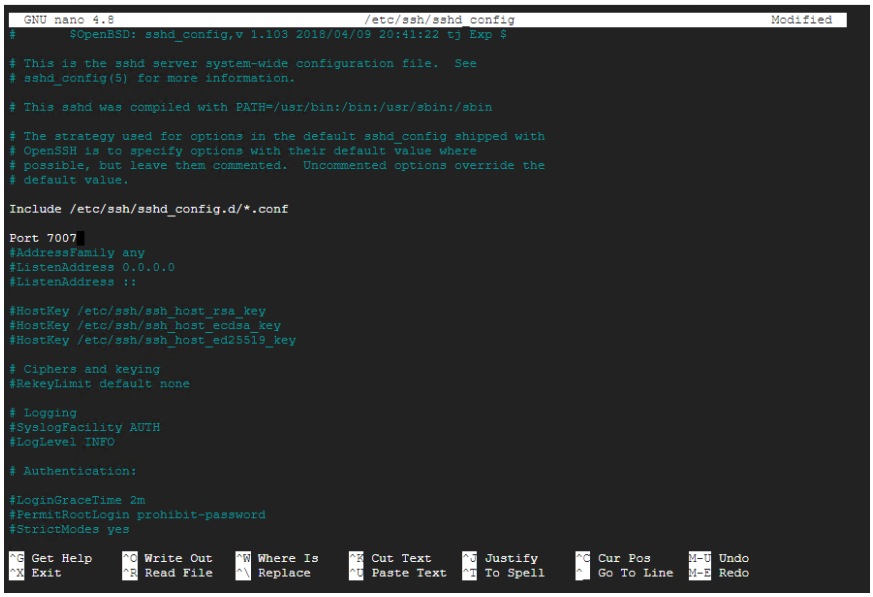
```
$ sudo ss -tulpn | grep ':7007'
```

```
coding4toon@james-bond-node:~$ sudo ss -tulpn | grep ':7007'
coding4toon@james-bond-node:~$
```

A blank response means the port is available.
A response in red means the port is in use. If so, choose another port.

Modify the SSH Configuration
Open the SSH configuration file:
$ sudo nano /etc/ssh/sshd_config

Locate the Port 22 line. If you don't see it, add it. Update the port number to your chosen value, like so:



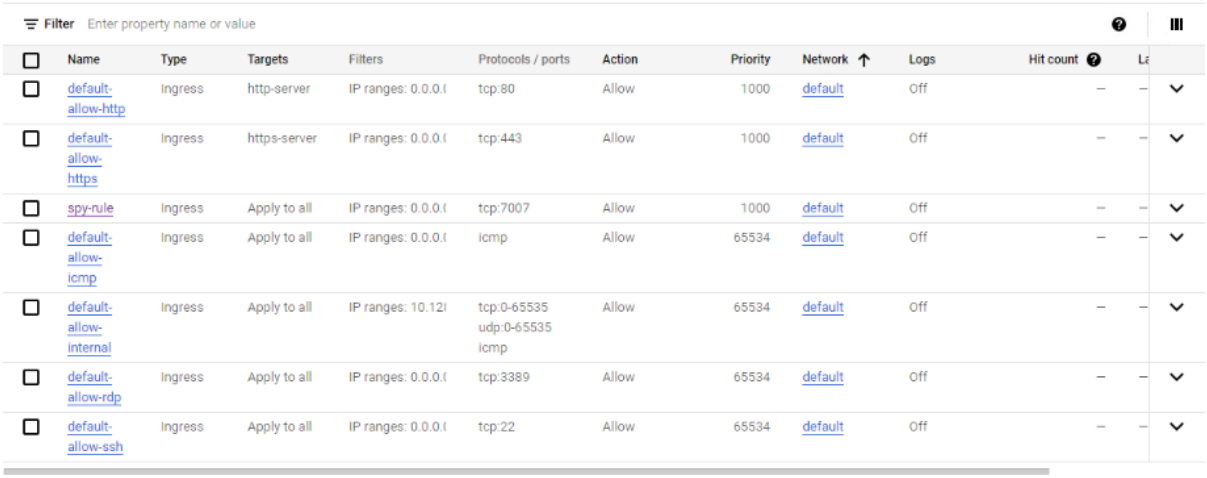
Apply the Changes
Restart the SSH service:
$ sudo systemctl restart ssh

```
coding4toon@james-bond-node:~$ sudo systemctl restart ssh
coding4toon@james-bond-node:~$
```

Update Your SSH Client
Remember to adjust your SSH client's settings or command to use the new port.

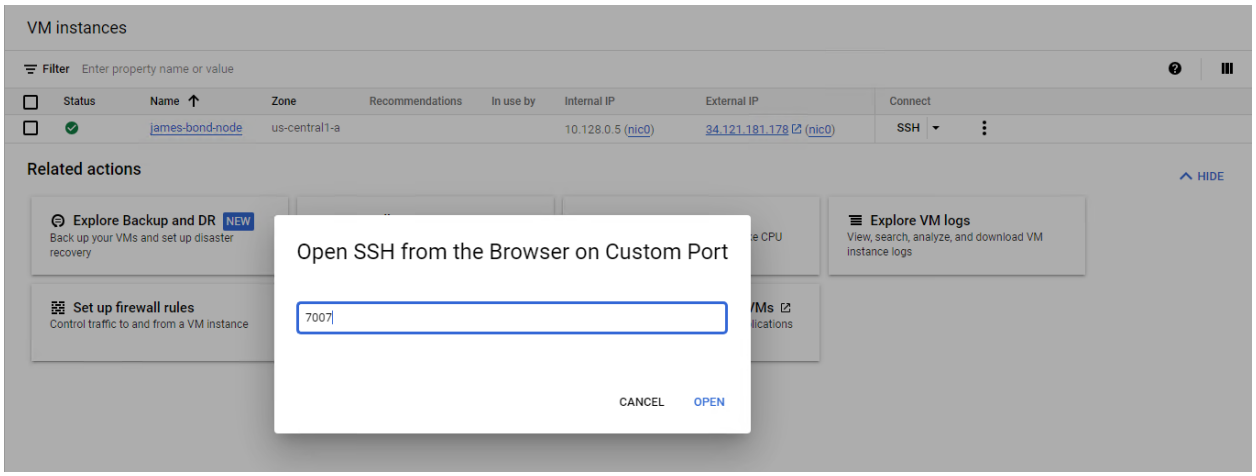
Google Cloud Platform Firewall Settings:
With the new SSH port, you need to allow traffic on this port in your Google Cloud Platform firewall rules.

- Go to Google Cloud Console.
- Navigate to “Firewall rules” and click on “Create firewall rule”.
- Name: e.g., "spy-node"
- Targets: "All instances in the network"
- Source IP ranges: 0.0.0.0/0, 35.235.240.0/20
- Protocols and ports: TCP and enter 7007
- Save the rule.



Test the New SSH Port

Log back in via SSH using new port (e.g. “7007”) to make sure everything is working correctly.



Agent T *- “By following these steps, you've made your server a bit safer from automated attacks. Always remember to use other security best practices too, like setting up a firewall, disabling root login, and using key-based authentication!”*

Setting Up UFW Firewall on Ubuntu 20.04

Agent T *- "UFW, or "Uncomplicated Firewall", is a user-friendly way to manage firewall settings on Ubuntu. It's designed to make setting up and maintaining a firewall easy without diving deep into iptables."*

Install UFW Firewall

If not already present, install the UFW firewall package:

$ sudo apt install ufw

```
coding4toon@james-bond-node:~$ sudo apt install ufw
Reading package lists... Done
Building dependency tree
Reading state information... Done
ufw is already the newest version (0.36-6ubuntu1.1).
ufw set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
coding4toon@james-bond-node:~$
```

Set Firewall Default Rules

To establish a secure default, deny all incoming connections and allow all outgoing connections.

$ sudo ufw default deny incoming

$ sudo ufw default allow outgoing

```
coding4toon@james-bond-node:~$ sudo ufw default deny incoming
Default incoming policy changed to 'deny'
(be sure to update your rules accordingly)
coding4toon@james-bond-node:~$ sudo ufw default allow outgoing
Default outgoing policy changed to 'allow'
(be sure to update your rules accordingly)
coding4toon@james-bond-node:~$
```

Secure Your SSH Connection

If you've changed the default SSH port (e.g., to 7007), allow inbound traffic on that port:

$ sudo ufw allow 7007/tcp

```
coding4toon@james-bond-node:~$ sudo ufw allow 7007/tcp
Rules updated
Rules updated (v6)
coding4toon@james-bond-node:~$
```

As a safety precaution, deny traffic on the old SSH port:

$ sudo ufw deny 22/tcp

```
coding4toon@james-bond-node:~$ sudo ufw deny 22/tcp
Rules updated
Rules updated (v6)
coding4toon@james-bond-node:~$
```

Open Ports for Various Services

For Ethereum's Execution Client:
$ sudo ufw allow 30303

```
coding4toon@james-bond-node:~$ sudo ufw allow 30303
Rules updated
Rules updated (v6)
coding4toon@james-bond-node:~$
```

For Ethereum's Consensus Client:
$ sudo ufw allow 8551/tcp

```
coding4toon@james-bond-node:~$ sudo ufw allow 8551/tcp
Rule added
Rule added (v6)
coding4toon@james-bond-node:~$
```

For Beacon Chain Node using Lighthouse:
$ sudo ufw allow 9000

```
coding4toon@james-bond-node:~$ sudo ufw allow 9000
Rules updated
Rules updated (v6)
coding4toon@james-bond-node:~$
```

For Grafana (a platform for monitoring and observability):
$ sudo ufw allow 3000/tcp

```
coding4toon@james-bond-node:~$ sudo ufw allow 3000/tcp
Rules updated
Rules updated (v6)
coding4toon@james-bond-node:~$
```

Activate and Check UFW

Once you've set up the rules, turn on UFW:
$ sudo ufw enable

```
coding4toon@james-bond-node:~$ sudo ufw enable
Command may disrupt existing ssh connections. Proceed with operation (y|n)? y
Firewall is active and enabled on system startup
coding4toon@james-bond-node:~$
```

Verify that the rules are correctly implemented:
$ sudo ufw status numbered

```
coding4toon@james-bond-node:~$ sudo ufw status numbered
Status: active

     To                         Action      From
     --                         ------      ----
[ 1] 7007/tcp                   ALLOW IN    Anywhere
[ 2] 22/tcp                     DENY IN     Anywhere
[ 3] 30303                      ALLOW IN    Anywhere
[ 4] 9000                       ALLOW IN    Anywhere
[ 5] 3000/tcp                   ALLOW IN    Anywhere
[ 6] 7007/tcp (v6)              ALLOW IN    Anywhere (v6)
[ 7] 22/tcp (v6)                DENY IN     Anywhere (v6)
[ 8] 30303 (v6)                 ALLOW IN    Anywhere (v6)
[ 9] 9000 (v6)                  ALLOW IN    Anywhere (v6)
[10] 3000/tcp (v6)              ALLOW IN    Anywhere (v6)

coding4toon@james-bond-node:~$
```

Test the Firewall
After configuring your firewall, it's essential to ensure you can still access your server. Log out and then SSH back into your server to make sure all is well.

Agent T *- "Nice, you've established a basic security layer for your Ubuntu 20.04 server with UFW. Keep in mind that security is multifaceted; always be on the lookout for additional measures to keep your systems secure."*

10.3- Configuring Swap Space and Timekeeping

Setting Up Swap Space on Ubuntu

Agent T *- "Think of swap space as an "emergency overflow" for your computer's memory. When the RAM gets full, the system starts using this disk space to keep things running, albeit more slowly."*

Check Existing Swap

Ensure you don't already have swap space set up:

$ free -h

```
coding4toon@james-bond-node:~$ free -h
               total        used        free      shared  buff/cache   available
Mem:            31Gi       343Mi        29Gi       0.0Ki       1.3Gi        30Gi
Swap:             0B          0B          0B
coding4toon@james-bond-node:~$
```

If you see zeros in the "Swap:" row, it means you don't have any swap space yet.

Agent T *- "How Much Swap Do You Need? Here's a guideline on the recommended swap space based on your RAM"*

RAM (GB)	Swap Size (GB)
8	3
12	3
16	4
24	5
32	6
64	8
128	11

Ensure You Have Disk Space

Check the available space:

$ df -h

```
coding4toon@james-bond-node:~$ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/root       243G  2.6G  240G   2% /
devtmpfs         16G     0   16G   0% /dev
tmpfs            16G     0   16G   0% /dev/shm
tmpfs           3.2G  968K  3.2G   1% /run
tmpfs           5.0M     0  5.0M   0% /run/lock
tmpfs            16G     0   16G   0% /sys/fs/cgroup
/dev/loop0       64M   64M     0 100% /snap/core20/1974
/dev/loop1      349M  349M     0 100% /snap/google-cloud-cli/163
/dev/loop2       54M   54M     0 100% /snap/snapd/19457
/dev/loop3       92M   92M     0 100% /snap/lxd/24061
/dev/sda15      105M  6.1M   99M   6% /boot/efi
tmpfs           3.2G     0  3.2G   0% /run/user/1001
coding4toon@james-bond-node:~$
```

Make sure there's enough space on the partition mounted as "/".

Creating the Swap Space

Based on your RAM, adjust the size accordingly. For instance, for a server with 32GB RAM:

$ sudo fallocate -l 6G /swapfile

$ sudo chmod 600 /swapfile

$ sudo mkswap /swapfile

$ sudo swapon /swapfile

```
coding4toon@james-bond-node:~$ sudo fallocate -l 6G /swapfile
coding4toon@james-bond-node:~$ sudo chmod 600 /swapfile
coding4toon@james-bond-node:~$ sudo mkswap /swapfile
Setting up swapspace version 1, size = 6 GiB (6442446848 bytes)
no label, UUID=61017b96-ded9-4d72-9150-f1da10fbc61b
coding4toon@james-bond-node:~$ sudo swapon /swapfile
coding4toon@james-bond-node:~$
```

Confirm the swap space has been created:

$ free -h

```
coding4toon@james-bond-node:~$ free -h
               total        used        free      shared  buff/cache   available
Mem:            31Gi       347Mi        29Gi       0.0Ki       1.3Gi        30Gi
Swap:          6.0Gi          0B       6.0Gi
coding4toon@james-bond-node:~$
```

Making the Swap Permanent

To ensure your swap remains active even after a reboot:

$ sudo cp /etc/fstab /etc/fstab.bak

$ echo '/swapfile none swap sw 0 0' | sudo tee -a /etc/fstab

```
coding4toon@james-bond-node:~$ sudo cp /etc/fstab /etc/fstab.bak
coding4toon@james-bond-node:~$ echo '/swapfile none swap sw 0 0' | sudo tee -a /etc/fstab
/swapfile none swap sw 0 0
coding4toon@james-bond-node:~$
```

Optimizing Swap Performance

Modify the system's swapping behavior for better performance

$ sudo sysctl vm.swappiness=10

$ sudo sysctl vm.vfs_cache_pressure=50

```
coding4toon@james-bond-node:~$ sudo sysctl vm.swappiness=10
vm.swappiness = 10
coding4toon@james-bond-node:~$ sudo sysctl vm.vfs_cache_pressure=50
vm.vfs_cache_pressure = 50
coding4toon@james-bond-node:~$
```

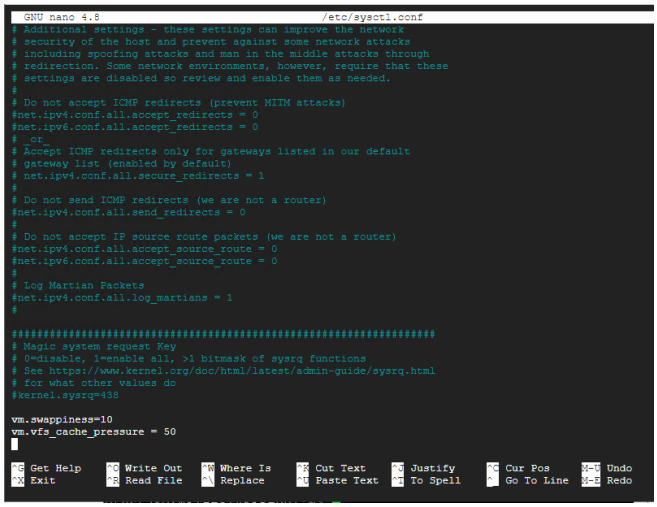
Make these changes permanent:

$ sudo nano /etc/sysctl.conf

At the end of the file, add:

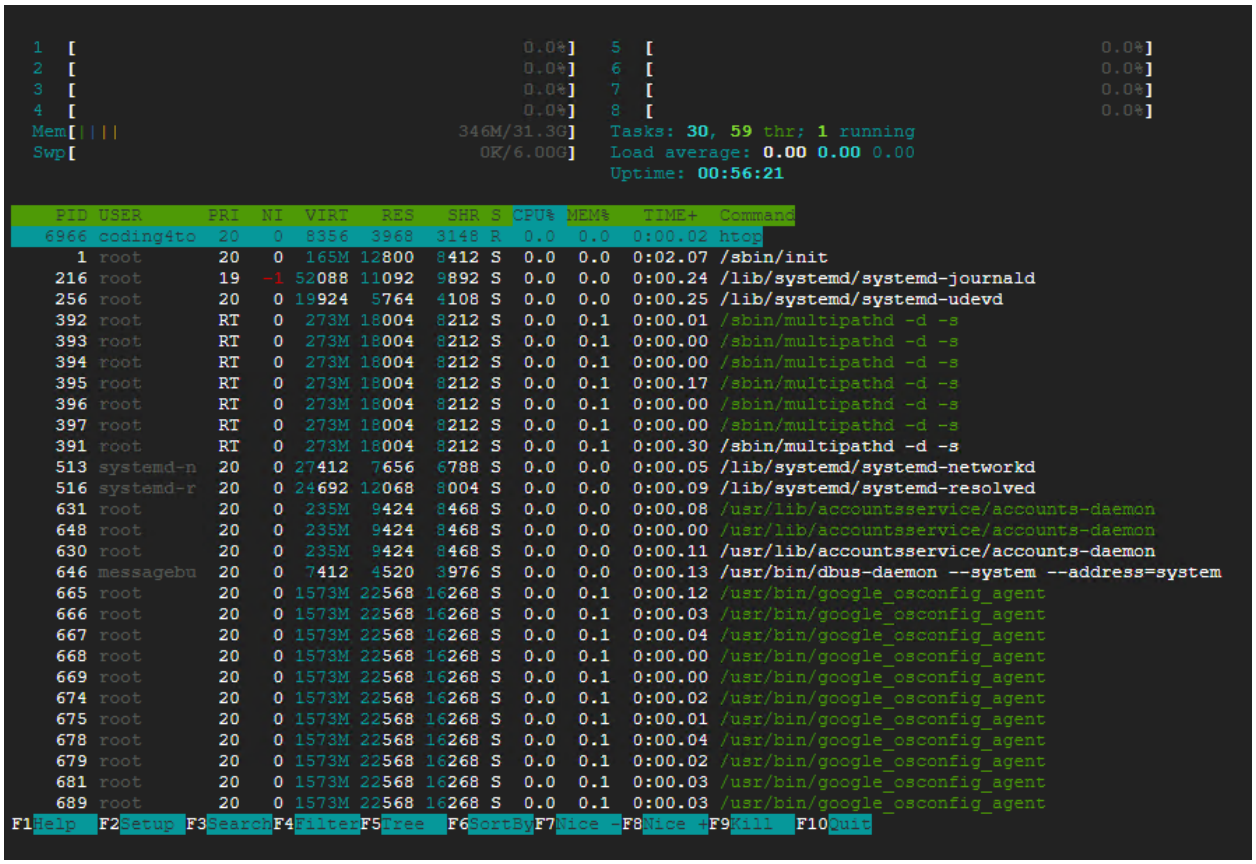
vm.swappiness=10

vm.vfs_cache_pressure = 50



Monitoring Your Swap

Use the htop command to monitor swap usage in real-time. If you don't have “htop”, you can install it with “$ sudo apt install htop”.



Agent T *- "Very nice, you've created and optimized a swap file for your Ubuntu system, enhancing its ability to handle memory-intensive tasks!".*

Configuring Timekeeping for Blockchain Validators on Ubuntu

Ubuntu has a built-in time synchronization mechanism via the timedatectl command, which is a part of the systemd system management daemon.

Agent T *- "Why Timekeeping Matters? For blockchain validators, precise timekeeping is crucial. A slight discrepancy in time can lead to issues with the blockchain's synchronization and, as a result, the validator's performance."*

Check the Current Timekeeping Status

To see the current status and ensure everything is set up correctly:

```
$ timedatectl
```

```
coding4toon@james-bond-node:~$ timedatectl
               Local time: Sun 2023-08-20 13:56:46 UTC
           Universal time: Sun 2023-08-20 13:56:46 UTC
                 RTC time: Sun 2023-08-20 13:56:46
                Time zone: Etc/UTC (UTC, +0000)
System clock synchronized: yes
              NTP service: active
          RTC in local TZ: no
coding4toon@james-bond-node:~$
```

Check the output. Ideally, the "NTP service" field should say "active" and the “System clock synchronized” field should say “yes”.

Google Cloud VMs typically have NTP set up out of the box to ensure accurate timekeeping.

Agent T *- “With these steps, you've ensured that your Ubuntu server, especially one on the Google Cloud Platform, maintains accurate timekeeping—vital for blockchain validator nodes and other time-sensitive operations. If ever you suspect time discrepancies, always recheck the NTP synchronization status using* ***timedatectl****.”*

10.4- Setting up JWT for Client-Client Communication on Your Server

Agent T *- "When your Execution and Consensus clients on the server communicate, it's vital to keep this exchange secure. That's where JWT comes in. By generating and using a JWT, you ensure that both clients can authenticate messages exchanged between them."*

Storing the JWT on Your Server

Begin by creating a specific directory to keep the JWT file secure:

$ sudo mkdir -p /var/lib/jwtsecret

```
coding4toon@james-bond-node:~$ sudo mkdir -p /var/lib/jwtsecret
coding4toon@james-bond-node:~$
```

Generating the JWT

You will generate a random 32-byte hex string which will serve as the JWT. For this, we'll use the openssl tool:

$ openssl rand -hex 32 | sudo tee /var/lib/jwtsecret/jwt.hex > /dev/null

```
coding4toon@james-bond-node:~$ openssl rand -hex 32 | sudo tee /var/lib/jwtsecret/jwt.hex > /dev/null
coding4toon@james-bond-node:~$
```

Checking the JWT

To view the content of the JWT file (the hex string) you just created:

$ sudo nano /var/lib/jwtsecret/jwt.hex

```
  GNU nano 4.8                                /var/lib/jwtsecret/jwt.hex
988da5152109e0bb74322f143417f2a2189bb7a6793830c1935ad86d2ce51500
```

Using the JWT in Client Configurations

As you move forward with setting up your clients, remember to reference the path to the jwt.hex file. This way, both the Execution and Consensus clients will use the JWT to authenticate messages they send or receive.

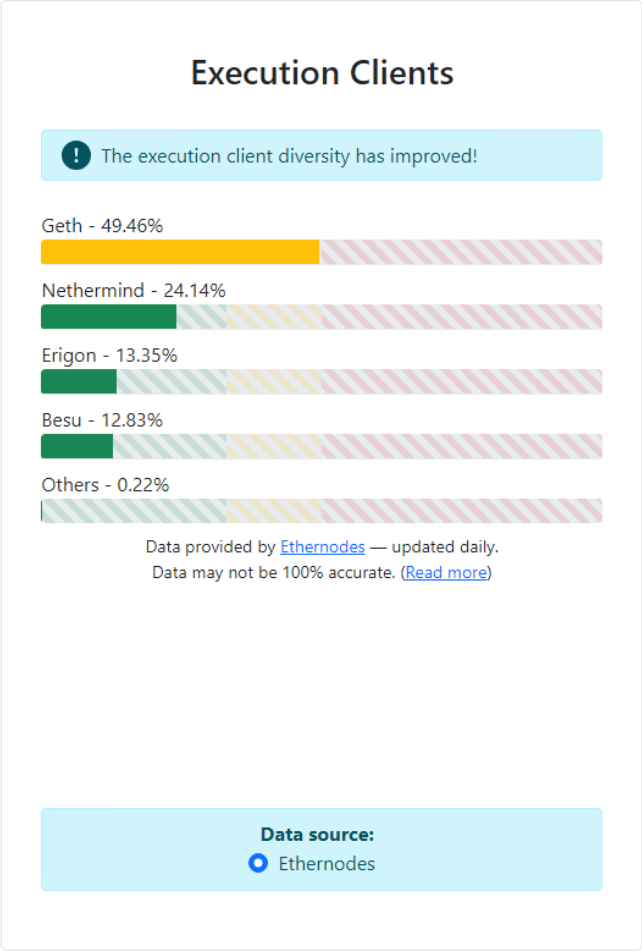
Agent T *- "By following these steps, you've successfully set up JWT-based authentication for inter-client communication on your server. This is a foundational step towards ensuring secure and authenticated communication between the Execution and Consensus clients."*

10.5- Setting-up an Execution Client - BEZU

Agent T *- "For staking on Ethereum, an Execution Client is essential. It's advised to select a minority client to promote client diversity and avoid potential penalties linked with bugs in majority clients."*

Checking Client Distribution

To see the current spread of clients in the network, check here. For instance, if Geth is the popular choice, think about selecting another client for diversity's sake.



Ressources

https://ethereum.org/en/developers/docs/nodes-and-clients/#execution-clients
https://clientdiversity.org/

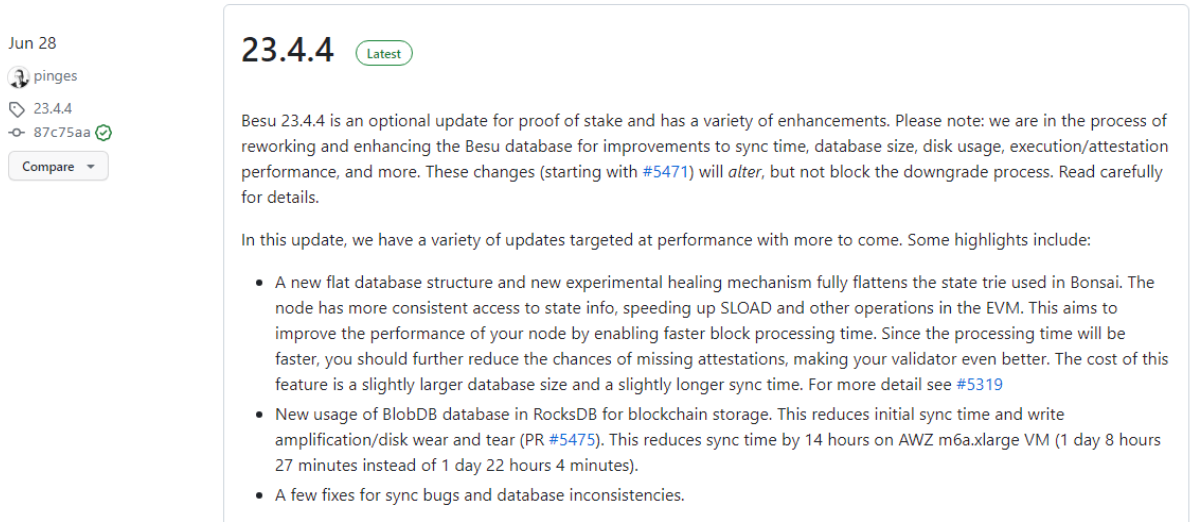
Agent T *- "This guide focuses on the Goerli test network, but ensuring client diversity is crucial for testing reasons. It offers you the chance to practice with a minority client before transitioning to the mainnet.Execution Clients need ample disk space because of Ethereum blockchain data storage."*

Installation of Hyperledger Besu

Hyperledger Besu is a viable Execution Client. You can install it by fetching the newest version. Under the "Download links", copy the link for the tar.gz file. Make sure you select the correct link.

Ressources
https://besu.hyperledger.org/
https://github.com/hyperledger/besu/releases

For instance:



Downloading and Setting Up Besu
Use the commands below to download, extract, and install Besu. Modify the URLs and filenames according to the version you're installing:
$ cd ~
$ curl -LO https://hyperledger.jfrog.io/artifactory/besu-binaries/besu/23.4.4/besu-23.4.4.tar.gz

```
coding4toon@james-bond-node:~$ cd ~
coding4toon@james-bond-node:~$ curl -LO https://hyperledger.jfrog.io/artifactory/besu-binaries/besu/23.4.4/besu
-23.4.4.tar.gz
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  166M  100  166M    0     0  30.2M      0  0:00:05  0:00:05 --:--:-- 34.0M
coding4toon@james-bond-node:~$
```

$ tar xvf besu-23.4.4.tar.gz

```
besu-23.4.4/lib/jnr-x86asm-1.0.2.jar
besu-23.4.4/lib/javax.persistence-api-2.2.jar
besu-23.4.4/lib/tuples-4.9.7.jar
besu-23.4.4/lib/grpc-context-1.53.0.jar
besu-23.4.4/lib/javax.annotation-api-1.3.2.jar
besu-23.4.4/lib/swagger-annotations-1.6.9.jar
besu-23.4.4/lib/bcutil-jdk18on-1.72.jar
besu-23.4.4/lib/bcprov-jdk18on-1.72.jar
besu-23.4.4/lib/jnr-constants-0.10.3.jar
besu-23.4.4/bin/
besu-23.4.4/bin/besu
besu-23.4.4/bin/besu.bat
besu-23.4.4/./
besu-23.4.4/./LICENSE
besu-23.4.4/./license-dependency.html
besu-23.4.4/./besu.autocomplete.sh
coding4toon@james-bond-node:~$
```

$ sudo cp -a besu-23.4.4 /usr/local/bin/besu

```
coding4toon@james-bond-node:~$ sudo cp -a besu-23.4.4 /usr/local/bin/besu
coding4toon@james-bond-node:~$
```

Clean up the files. Modify the file name to match the downloaded version.

$ rm besu-23.4.4.tar.gz

$ rm -r besu-23.4.4

```
coding4toon@james-bond-node:~$ rm besu-23.4.4.tar.gz
coding4toon@james-bond-node:~$ rm -r besu-23.4.4
coding4toon@james-bond-node:~$
```

Installing Necessary Prerequisites

For Besu to run correctly, some prerequisites are required. Install them with:

$ sudo apt -y install default-jre

```
Selecting previously unselected package default-jre.
Preparing to unpack .../default-jre_2%3a1.11-72_amd64.deb ...
Unpacking default-jre (2:1.11-72) ...
Setting up openjdk-11-jre-headless:amd64 (11.0.20+8-1ubuntu1~20.04) ...
update-alternatives: using /usr/lib/jvm/java-11-openjdk-amd64/bin/jjs to provide /usr/bin/jjs (jjs) in auto mod
e
update-alternatives: using /usr/lib/jvm/java-11-openjdk-amd64/bin/rmid to provide /usr/bin/rmid (rmid) in auto
mode
update-alternatives: using /usr/lib/jvm/java-11-openjdk-amd64/bin/pack200 to provide /usr/bin/pack200 (pack200)
 in auto mode
update-alternatives: using /usr/lib/jvm/java-11-openjdk-amd64/bin/unpack200 to provide /usr/bin/unpack200 (unpa
ck200) in auto mode
Setting up openjdk-11-jre:amd64 (11.0.20+8-1ubuntu1~20.04) ...
Setting up default-jre-headless (2:1.11-72) ...
Setting up default-jre (2:1.11-72) ...
Processing triggers for hicolor-icon-theme (0.17-2) ...
Processing triggers for mime-support (3.64ubuntu1) ...
coding4toon@james-bond-node:~$
```

$ sudo apt install -y libjemalloc-dev

```
235 kB]
Get:2 http://us-central1.gce.archive.ubuntu.com/ubuntu focal/universe amd64 libjemalloc-dev amd64 5.2.1-1ubuntu
1 [425 kB]
Fetched 660 kB in 0s (7221 kB/s)
Selecting previously unselected package libjemalloc2:amd64.
(Reading database ... 78151 files and directories currently installed.)
Preparing to unpack .../libjemalloc2_5.2.1-1ubuntu1_amd64.deb ...
Unpacking libjemalloc2:amd64 (5.2.1-1ubuntu1) ...
Selecting previously unselected package libjemalloc-dev.
Preparing to unpack .../libjemalloc-dev_5.2.1-1ubuntu1_amd64.deb ...
Unpacking libjemalloc-dev (5.2.1-1ubuntu1) ...
Setting up libjemalloc2:amd64 (5.2.1-1ubuntu1) ...
Setting up libjemalloc-dev (5.2.1-1ubuntu1) ...
Processing triggers for man-db (2.9.1-1) ...
Processing triggers for libc-bin (2.31-0ubuntu9.9) ...
coding4toon@james-bond-node:~$
```

Agent T *- "By following the steps above, you've successfully set up the Execution Client, specifically Hyperledger Besu, and are one step closer to staking on Ethereum. Remember, diversity is essential in the client ecosystem, so your choice matters!"*

Setting up Besu as a Background Service

Create a Besu User

This will be a system user specifically for running Besu, which cannot log into the server.

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

```
coding4toon@james-bond-node:~$ sudo useradd --no-create-home --shell /bin/false besu
coding4toon@james-bond-node:~$
```

Create Data Directory

Besu needs a directory to store Ethereum blockchain data.

$ sudo mkdir -p /var/lib/besu

```
coding4toon@james-bond-node:~$ sudo mkdir -p /var/lib/besu
coding4toon@james-bond-node:~$
```

Assign Permissions

Give the besu user account permissions to modify the data directory.

$ sudo chown -R besu:besu /var/lib/besu

```
coding4toon@james-bond-node:~$ sudo chown -R besu:besu /var/lib/besu
coding4toon@james-bond-node:~$
```

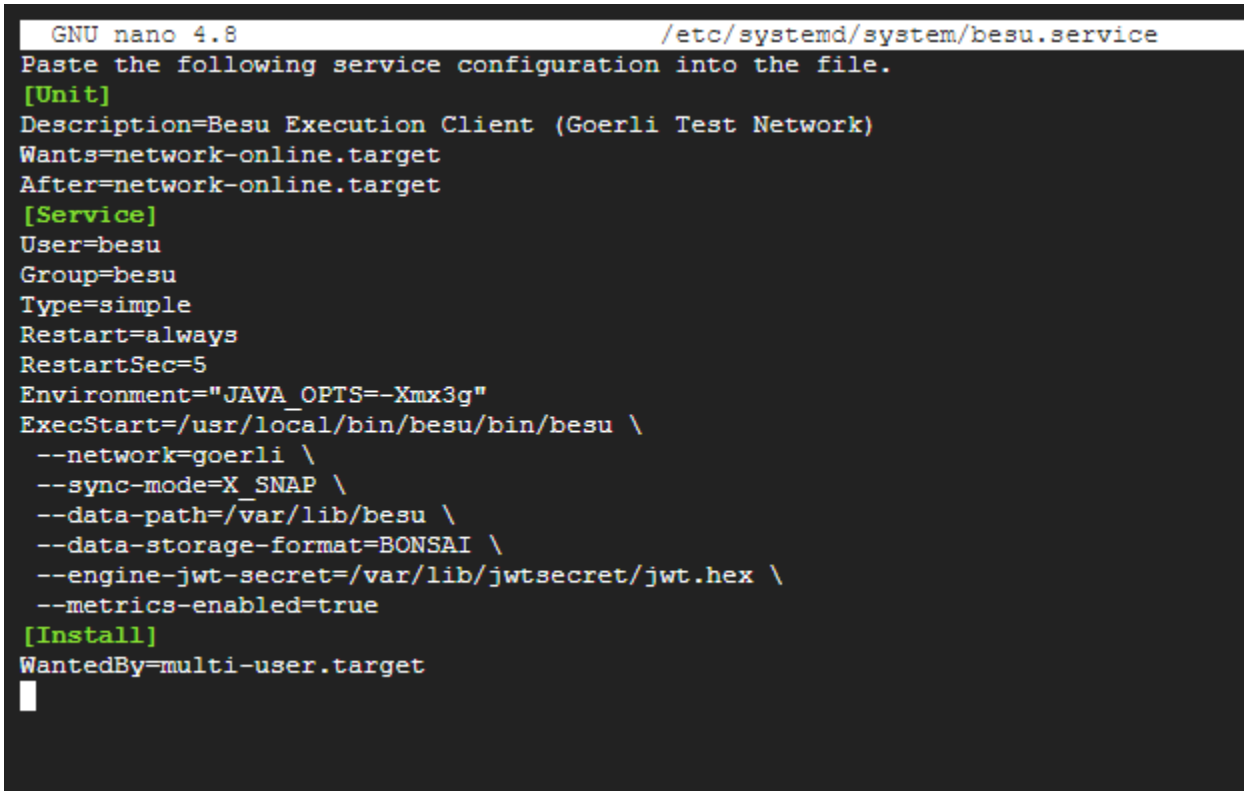
Systemd Service Configuration

Start by creating the systemd service config file.

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

Inside the file, add the following service configuration:

```
[Unit]
Description=Besu Execution Client (Goerli Test Network)
Wants=network-online.target
After=network-online.target
[Service]
User=besu
Group=besu
Type=simple
Restart=always
RestartSec=5
Environment="JAVA_OPTS=-Xmx3g"
ExecStart=/usr/local/bin/besu/bin/besu \
 --network=goerli \
 --sync-mode=X_SNAP \
 --data-path=/var/lib/besu \
 --data-storage-format=BONSAI \
 --engine-jwt-secret=/var/lib/jwtsecret/jwt.hex \
 --metrics-enabled=true
[Install]
WantedBy=multi-user.target
```



Explanation of Key Flags

- JAVA_OPTS=-Xmx3g: Defines the max memory the Besu process can use, set here to 3GB.
- --sync-mode=X_SNAP: The recommended sync mode with the BONSAI data storage format.
- --data-storage-format=BONSAI: An efficient way to manage blockchain data.
- --engine-jwt-secret: Points to the authentication token for communication between the Execution and Consensus clients.
- --metrics-enabled: Enables output for Prometheus monitoring.

Starting the Besu Service

After setting up the configuration, make systemd aware of it and start Besu.

$ sudo systemctl daemon-reload

$ sudo systemctl start besu

$ sudo systemctl status besu

```
coding4toon@james-bond-node:~$ sudo systemctl daemon-reload
coding4toon@james-bond-node:~$ sudo systemctl start besu
coding4toon@james-bond-node:~$ sudo systemctl status besu
● besu.service - Besu Execution Client (Goerli Test Network)
     Loaded: loaded (/etc/systemd/system/besu.service; disabled; vendor preset: enabled)
     Active: active (running) since Sun 2023-08-20 14:25:44 UTC; 8s ago
   Main PID: 7756 (java)
      Tasks: 74 (limit: 38496)
     Memory: 369.2M
     CGroup: /system.slice/besu.service
             └─7756 java -Dvertx.disableFileCPResolving=true -Dbesu.home=/usr/local/bin/besu -Dlog4j.shutdownH>

Aug 20 14:25:47 james-bond-node besu[7756]: 2023-08-20 14:25:47.983+00:00 | vert.x-eventloop-thread-1 | INFO  >
Aug 20 14:25:47 james-bond-node besu[7756]: 2023-08-20 14:25:47.984+00:00 | vert.x-eventloop-thread-1 | INFO  >
Aug 20 14:25:48 james-bond-node besu[7756]: 2023-08-20 14:25:48.015+00:00 | vert.x-eventloop-thread-1 | INFO  >
Aug 20 14:25:48 james-bond-node besu[7756]: 2023-08-20 14:25:48.051+00:00 | main | INFO  | DefaultP2PNetwork |>
Aug 20 14:25:48 james-bond-node besu[7756]: 2023-08-20 14:25:48.052+00:00 | main | INFO  | DefaultP2PNetwork |>
Aug 20 14:25:48 james-bond-node besu[7756]: 2023-08-20 14:25:48.062+00:00 | main | INFO  | NetworkRunner | Sup>
Aug 20 14:25:48 james-bond-node besu[7756]: 2023-08-20 14:25:48.063+00:00 | main | INFO  | DefaultSynchronizer>
Aug 20 14:25:48 james-bond-node besu[7756]: 2023-08-20 14:25:48.064+00:00 | main | INFO  | FastSyncDownloader >
Aug 20 14:25:48 james-bond-node besu[7756]: 2023-08-20 14:25:48.068+00:00 | main | INFO  | Runner | Ethereum m>
Aug 20 14:25:52 james-bond-node systemd[1]: /etc/systemd/system/besu.service:1: Assignment outside of section.>
lines 1-19/19 (END)
```

Check for active (running) in green text. If there's an issue, review your steps.
To exit, press Q.

Monitoring Besu's Output

You can track Besu's progress and see any potential errors.

$ sudo journalctl -fu besu

```
coding4toon@james-bond-node:~$ sudo journalctl -fu besu
-- Logs begin at Sun 2023-08-20 12:55:08 UTC. --
Aug 20 14:25:47 james-bond-node besu[7756]: 2023-08-20 14:25:47.984+00:00 | vert.x-eventloop-thread-1 | INFO  |
 PeerDiscoveryAgent | P2P peer discovery agent started and listening on /[0:0:0:0:0:0:0:0%0]:30303
Aug 20 14:25:48 james-bond-node besu[7756]: 2023-08-20 14:25:48.015+00:00 | vert.x-eventloop-thread-1 | INFO  |
 PeerDiscoveryAgent | Writing node record to disk. NodeRecord{seq=1, publicKey=0x03bf9289cba2ecff4adafcbe74d314
92a4d59e60376d9a1268ed53f8faf243ba02, udpAddress=Optional[/127.0.0.1:30303], tcpAddress=Optional[/127.0.0.1:303
03], asBase64=-Jq4QCcKsYp0X5oUZ5Z7Jw4hdJ3tggplNCbL2VyEeM3Xltl0I_xOvpY_1w0Y8zJ9qKk5z7OarjatBsas_1yH093q8qABg2V0a
MrJhKP1qwiDF9QzgmlkgnY0gmlwhH8AAAGJc2VjcDI1NmsxoQO_konLouz_Str8vnTTFJKk1Z5gN22aEmjtU_j68kO6AoN0Y3CCdl-DdWRwgnZf
, nodeId=0x1821020719befddd2e671980c756e8a7c6c66e62eeaae2ea570050c5e4c00bc7, customFields={tcp=30303, udp=30303
, ip=0x7f000001, eth=[[0xa3f5ab08, 0x17d433]], id=V4, secp256k1=0x03bf9289cba2ecff4adafcbe74d31492a4d59e60376d9
a1268ed53f8faf243ba02}}
Aug 20 14:25:48 james-bond-node besu[7756]: 2023-08-20 14:25:48.051+00:00 | main | INFO  | DefaultP2PNetwork |
Enode URL enode://bf9289cba2ecff4adafcbe74d31492a4d59e60376d9a1268ed53f8faf243ba02308242d0619cfdcc233fd6faf44f9
a01cd55b2d743a9ef7d4542ca25a86ef1b5@127.0.0.1:30303
Aug 20 14:25:48 james-bond-node besu[7756]: 2023-08-20 14:25:48.052+00:00 | main | INFO  | DefaultP2PNetwork |
Node address 0xc756e8a7c6c66e62eeaae2ea570050c5e4c00bc7
Aug 20 14:25:48 james-bond-node besu[7756]: 2023-08-20 14:25:48.062+00:00 | main | INFO  | NetworkRunner | Supp
orted capabilities: [eth/63, eth/64, eth/65, eth/66, eth/67, eth/68], [snap/1]
Aug 20 14:25:48 james-bond-node besu[7756]: 2023-08-20 14:25:48.063+00:00 | main | INFO  | DefaultSynchronizer
| Starting synchronizer.
Aug 20 14:25:48 james-bond-node besu[7756]: 2023-08-20 14:25:48.064+00:00 | main | INFO  | FastSyncDownloader |
 Starting sync
Aug 20 14:25:48 james-bond-node besu[7756]: 2023-08-20 14:25:48.068+00:00 | main | INFO  | Runner | Ethereum ma
in loop is up.
Aug 20 14:25:52 james-bond-node systemd[1]: /etc/systemd/system/besu.service:1: Assignment outside of section.
Ignoring.
Aug 20 14:25:56 james-bond-node besu[7756]: 2023-08-20 14:25:56.453+00:00 | Timer-0 | INFO  | DNSResolver | Res
olved 426 nodes
Aug 20 14:26:48 james-bond-node besu[7756]: 2023-08-20 14:26:48.069+00:00 | EthScheduler-Timer-0 | INFO  | Pivo
tSelectorFromSafeBlock | Waiting for consensus client, this may be because your consensus client is still synci
ng
```

To exit, press CTRL + C.

Autostart on Reboot

Ensure Besu automatically starts up after any system reboots.

$ sudo systemctl enable besu

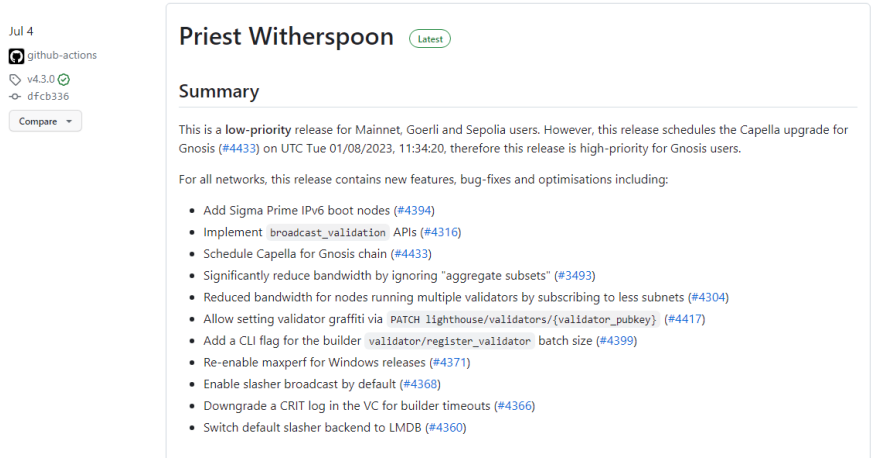
```
coding4toon@james-bond-node:~$ sudo systemctl enable besu
Created symlink /etc/systemd/system/multi-user.target.wants/besu.service → /etc/systemd/system/besu.service.
coding4toon@james-bond-node:~$
```

Agent T - *"Wonderful,* with the above steps, Besu will be set up as a background service and will start syncing. Remember, it will sync but will await the consensus client configuration for completion."

10.6- Setting Up the Consensus Client - LIGHTHOUSE

Agent T *- "The Lighthouse Consensus Client is a single binary which encapsulates the functionality of the beacon node and validator, run in separate instances."*

Identifying the Latest Release
Head over to the Lighthouse release page to find the latest release.



Ressources
https://github.com/sigp/lighthouse/releases
https://lighthouse-book.sigmaprime.io/installation-binaries.html

Copying the Binary
From the "Binaries" section, make sure to copy the download link of the correct version, for instance: lighthouse-v4.3.0-x86_64-unknown-linux-gnu.tar.gz.

Agent T *- "There are two types of Lighthouse binaries — portable and non-portable. While the portable version is more universally compatible, the non-portable version offers better performance (about 20-30% speed up) but has specific hardware requirements. As Google Cloud Platform provides modern CPUs then it is possible to run a non-portable build to get a 20-30% speed up."*

Check Compatibility on Google Cloud Platform (GCP)
On the GCP dashboard, click on your VM instance (like "james-bond-node") to see its details. Look at the "Machine configuration" section. If GCP lists a CPU like "Intel Cascade Lake (2019)", it indicates compatibility with the optimized build of Lighthouse.

Machine configuration

Machine type	n2-standard-8
CPU platform	Intel Cascade Lake
Minimum CPU platform	None
Architecture	x86/64

Download the Binary

Navigate to your home directory and download the Lighthouse archive.

$ cd ~

$ curl -LO https://github.com/sigp/lighthouse/releases/download/v4.3.0/lighthouse-v4.3.0-x86_64-unknown-linux-gnu.tar.gz

```
coding4toon@james-bond-node:~$ cd ~
coding4toon@james-bond-node:~$ curl -LO https://github.com/sigp/lighthouse/releases/download/v4.3.0/lighthouse-
v4.3.0-x86_64-unknown-linux-gnu.tar.gz
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100 39.0M  100 39.0M    0     0  76.3M      0 --:--:-- --:--:-- --:--:--  109M
coding4toon@james-bond-node:~$
```

Extract and Move

Extract the binary from the archive and move it to the /usr/local/bin directory so it can be executed from anywhere in the terminal.

$ tar xvf lighthouse-v4.3.0-x86_64-unknown-linux-gnu.tar.gz

$ sudo cp lighthouse /usr/local/bin

```
coding4toon@james-bond-node:~$ tar xvf lighthouse-v4.3.0-x86_64-unknown-linux-gnu.tar.gz
lighthouse
coding4toon@james-bond-node:~$ sudo cp lighthouse /usr/local/bin
coding4toon@james-bond-node:~$
```

Cleanup

Delete the downloaded archive and the extracted binary file from your home directory to save space.

$ rm lighthouse-v4.3.0-x86_64-unknown-linux-gnu.tar.gz

$ rm lighthouse

```
coding4toon@james-bond-node:~$ rm lighthouse-v4.3.0-x86_64-unknown-linux-gnu.tar.gz
coding4toon@james-bond-node:~$ rm lighthouse
coding4toon@james-bond-node:~$
```

Agent T *- "You have successfully downloaded and set up the Lighthouse Consensus Client on your server!"*

10.7- Generating Staking Data

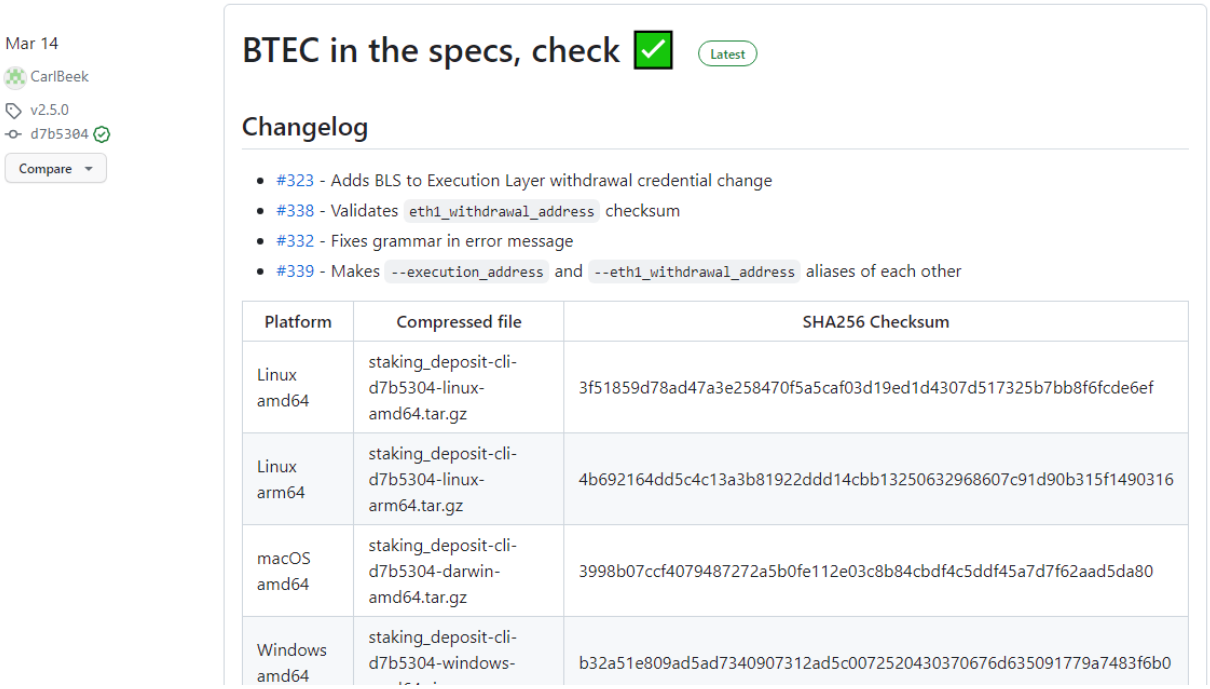
Setting Up the Staking Deposit Tool on Goerli Test Network

To participate in staking on the Goerli test network, you need to use the deposit tool to generate validator data files. Each validator requires 32 Goerli test network ETH for activation.

Agent T *- "The amount of Goerli test network ETH available per person is limited. We will use* ***EthStaker Goerli Bot*** *to fund our validator node."*

Downloading the Deposit Tool (Staking Deposit CLI)

First, head over to the staking deposit CLI tool's release page to identify the latest version.



Mar 14
CarlBeek
v2.5.0
d7b5304
Compare

BTEC in the specs, check ✅ Latest

Changelog

- #323 - Adds BLS to Execution Layer withdrawal credential change
- #338 - Validates `eth1_withdrawal_address` checksum
- #332 - Fixes grammar in error message
- #339 - Makes `--execution_address` and `--eth1_withdrawal_address` aliases of each other

Platform	Compressed file	SHA256 Checksum
Linux amd64	staking_deposit-cli-d7b5304-linux-amd64.tar.gz	3f51859d78ad47a3e258470f5a5caf03d19ed1d4307d517325b7bb8f6fcde6ef
Linux arm64	staking_deposit-cli-d7b5304-linux-arm64.tar.gz	4b692164dd5c4c13a3b81922ddd14cbb13250632968607c91d90b315f1490316
macOS amd64	staking_deposit-cli-d7b5304-darwin-amd64.tar.gz	3998b07ccf4079487272a5b0fe112e03c8b84cbdf4c5ddf45a7d7f62aad5da80
Windows amd64	staking_deposit-cli-d7b5304-windows-	b32a51e809ad5ad7340907312ad5c0072520430370676d635091779a7483f6b0

Ressources

https://github.com/ethereum/staking-deposit-cli/releases

Identify the VM Architecture

You can determine the architecture of your virtual machine in two ways:

1- Via the terminal using the command:

$ dpkg –print-architecture

```
coding4toon@james-bond-node:~$ dpkg --print-architecture
amd64
coding4toon@james-bond-node:~$
```

2- Or, by accessing your VM instance's details on the Google Cloud Platform dashboard and checking the "Boot Disk" section.

Boot disk

Name	Image	Interface type	Size (GB)	Device name	Type	Architecture	Encryption
james-bond-node	ubuntu-2004-focal-v20230817	SCSI	250	james-bond-node	Balanced persistent disk	x86/64	Google-managed

Downloading the Archive

Navigate to your home directory and download the deposit tool archive. Be sure to modify the URL to match the correct and latest version.

$ cd ~

$ curl -LO https://github.com/ethereum/staking-deposit-cli/releases/download/v2.5.0/staking_deposit-cli-d7b5304-linux-amd64.tar.gz

```
coding4toon@james-bond-node:~$ cd ~
coding4toon@james-bond-node:~$ curl -LO https://github.com/ethereum/staking-deposit-cli/releases/download/v2.5.
0/staking_deposit-cli-d7b5304-linux-amd64.tar.gz
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100 20.3M  100 20.3M    0     0  47.8M      0 --:--:-- --:--:-- --:--:-- 47.8M
coding4toon@james-bond-node:~$
```

Unpacking and Accessing the Directory

Extract the contents of the archive and navigate to the directory it creates. Again, ensure you modify the file name to match the version you downloaded.

$ tar xvf staking_deposit-cli-d7b5304-linux-amd64.tar.gz

$ cd staking_deposit-cli-d7b5304-linux-amd64

```
coding4toon@james-bond-node:~$ tar xvf staking_deposit-cli-d7b5304-linux-amd64.tar.gz
./staking_deposit-cli-d7b5304-linux-amd64/
./staking_deposit-cli-d7b5304-linux-amd64/deposit
coding4toon@james-bond-node:~$ cd staking_deposit-cli-d7b5304-linux-amd64
coding4toon@james-bond-node:~/staking_deposit-cli-d7b5304-linux-amd64$
```

Checking the Binary File

Inside the directory, you should find an executable named deposit.

```
coding4toon@james-bond-node:~/staking_deposit-cli-d7b5304-linux-amd64$ ls
deposit
coding4toon@james-bond-node:~/staking_deposit-cli-d7b5304-linux-amd64$
```

 ***Agent T*** *- "You have successfully set up the staking deposit tool! This will help you generate the necessary data files to stake on the Goerli test network."*

Preparing to Run the Staking Deposit Tool (CLI)

The deposit tool will generate a mnemonic key. While we're on a test network and the stakes are low, it's crucial to practice good security habits. Always handle this key securely and make sure it doesn't get exposed.

 ***Agent T*** *- "As you're working on GCP with the appropriate firewall rules, the current machine should be considered secure, allowing you to run the deposit tool directly from it without worries."*

Running the Staking Deposit Tool (CLI)

Command Execution
On your secure machine, execute the binary file of the deposit tool with the following command:
$ ./deposit new-mnemonic --num_validators <numberOfValidator> --chain goerli --eth1_withdrawal_address <YourWithdrawalAaddress>

- Replace <numberOfValidator> with the number of validators you want to set up.
- Replace <YourWithdrawalAddress> with an Ethereum address on the Goerli test network that you control. If you're short on Goerli ETH, you can use the EthStaker Goerli Bot's address.

EthStaker Goerli Bot Address
As you need Goerli test ETH and wish to use the **EthStaker Goerli Bot** for it, your command would look like this:

$ ./deposit new-mnemonic --num_validators 1 --chain goerli --eth1_withdrawal_address 0x4D496CcC28058B1D74B7a19541663E21154f9c84

```
coding4toon@james-bond-node:~/staking_deposit-cli-d7b5304-linux-amd64$ ./deposit new-mnemonic --num_validators
1 --chain goerli --eth1_withdrawal_address 0x4D496CcC28058B1D74B7a19541663E21154f9c84

***Using the tool on an offline and secure device is highly recommended to keep your mnemonic safe.***

Please choose your language ['1. العربية', '2. ελληνικά', '3. English', '4. Français', '5. Bahasa melayu', '6.
Italiano', '7. 日本語', '8. 한국어', '9. Português do Brasil', '10. român', '11. Türkçe', '12. 简体中文']:  [En
glish]:
```

```
coding4toon@james-bond-node:~/staking_deposit-cli-d7b5304-linux-amd64$ ./deposit new-mnemonic --num_validators
1 --chain goerli --eth1_withdrawal_address 0x4D496CcC28058B1D74B7a19541663E21154f9c84

***Using the tool on an offline and secure device is highly recommended to keep your mnemonic safe.***

Please choose your language ['1. العربية', '2. ελληνικά', '3. English', '4. Français', '5. Bahasa melayu', '6.
Italiano', '7. 日本語', '8. 한국어', '9. Português do Brasil', '10. român', '11. Türkçe', '12. 简体中文']:  [En
glish]: 3

**[Warning] you are setting an Eth1 address as your withdrawal address. Please ensure that you have control ove
r this address.**

Repeat your execution address for confirmation.:
```

EthStaker Goerli Bot address : 0x4D496CcC28058B1D74B7a19541663E21154f9c84

Withdrawal Address Importance

Once you've set the withdrawal address, it can't be changed later. Be absolutely sure about the address you specify, ensuring it's an address you have full control over or the **EthStaker Goerli Bot** address. If you ever decide to go mainnet, always use an address you own.

The --eth1_withdrawal_address Flag

This flag lets you define a Goerli test Ethereum address where any staking rewards beyond the 32 staked Goerli test ETH will be sent to (once withdrawals are allowed). If you leave your validator position, this will also be the address where your initial 32 staked Goerli test ETH is sent to.

Not Setting the --eth1_withdrawal_address Flag

If you don’t specify this flag now, you'll have an opportunity to set it later through a specific procedure. If you skip this, automatic withdrawals of staking rewards won't happen, and you can't get back your 32 staked Goerli test ETH after exiting the validator until you adjust the withdrawal credentials.

Agent T *- “By adhering to these steps and bearing in mind the notes, you will securely and effectively set up and use the staking deposit tool on the Goerli test network.”*

Setting Validator Keystore Password

When you execute the mentioned steps and specify your preferred language, you'll be prompted to create a password for the validator keystore.

Agent T *- "Ensure you store this password safely and securely. It will be essential when you're loading the validator keys into the Consensus Client validator wallet later."*

```
coding4toon@james-bond-node:~/staking_deposit-cli-d7b5304-linux-amd64$ ./deposit new-mnemonic --num_validators
1 --chain goerli --eth1_withdrawal_address 0x4D496CcC28058B1D74B7a19541663E21154f9c84

***Using the tool on an offline and secure device is highly recommended to keep your mnemonic safe.***

Please choose your language ['1. العربية', '2. ελληνικά', '3. English', '4. Français', '5. Bahasa melayu', '6.
Italiano', '7. 日本語', '8. 한국어', '9. Português do Brasil', '10. român', '11. Türkçe', '12. 简体中文']:  [En
glish]: 3

**[Warning] you are setting an Eth1 address as your withdrawal address. Please ensure that you have control ove
r this address.**

Repeat your execution address for confirmation.: 0x4D496CcC28058B1D74B7a19541663E21154f9c84

**[Warning] you are setting an Eth1 address as your withdrawal address. Please ensure that you have control ove
r this address.**

Please choose the language of the mnemonic word list ['1. 简体中文', '2. 繁體中文', '3. čeština', '4. English',
 '5. Italiano', '6. 한국어', '7. Português', '8. Español']:  [english]:
```

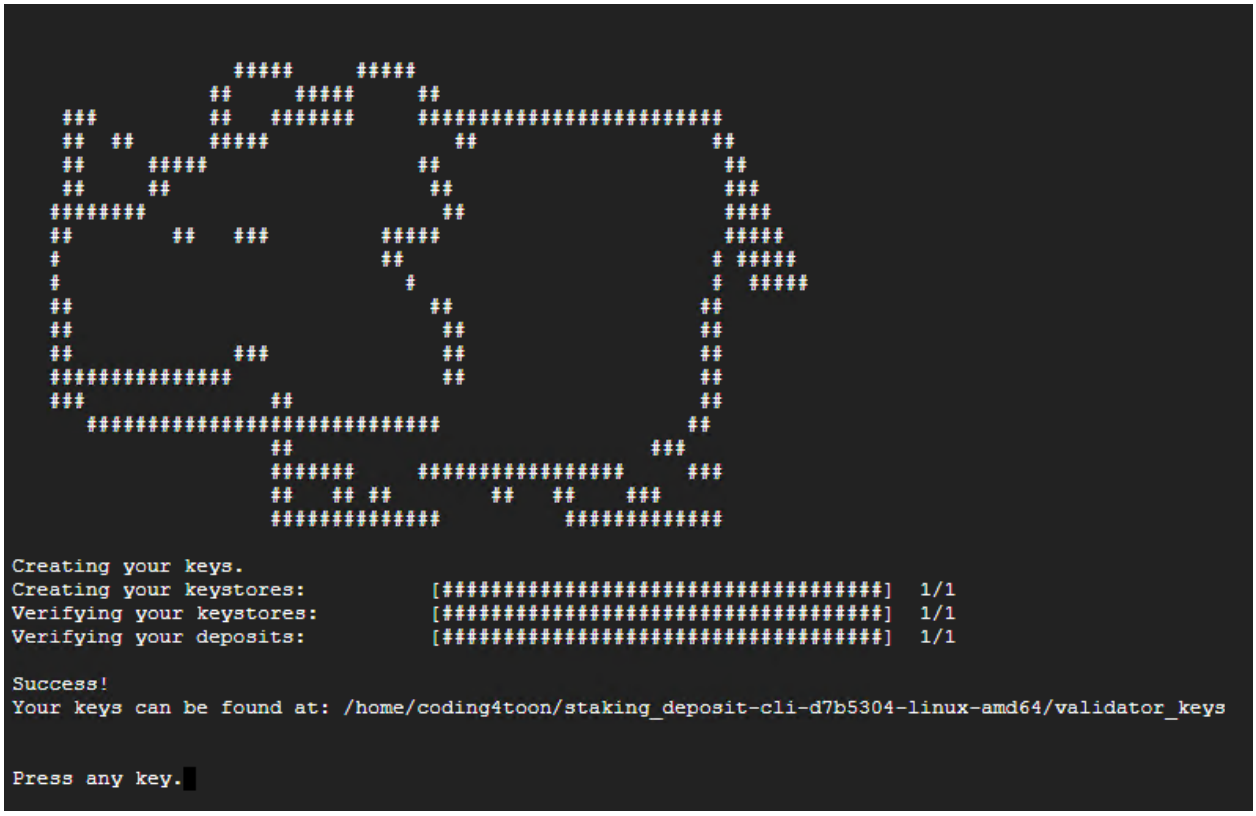
```
coding4toon@james-bond-node:~/staking_deposit-cli-d7b5304-linux-amd64$ ./deposit new-mnemonic --num_validators
1 --chain goerli --eth1_withdrawal_address 0x4D496CcC28058B1D74B7a19541663E21154f9c84

***Using the tool on an offline and secure device is highly recommended to keep your mnemonic safe.***

Please choose your language ['1. العربية', '2. ελληνικά', '3. English', '4. Français', '5. Bahasa melayu', '6.
Italiano', '7. 日本語', '8. 한국어', '9. Português do Brasil', '10. român', '11. Türkçe', '12. 简体中文']:  [En
glish]: 3

**[Warning] you are setting an Eth1 address as your withdrawal address. Please ensure that you have control ove
r this address.**

Repeat your execution address for confirmation.: 0x4D496CcC28058B1D74B7a19541663E21154f9c84

**[Warning] you are setting an Eth1 address as your withdrawal address. Please ensure that you have control ove
r this address.**

Please choose the language of the mnemonic word list ['1. 简体中文', '2. 繁體中文', '3. čeština', '4. English',
 '5. Italiano', '6. 한국어', '7. Português', '8. Español']:  [english]: 4
Create a password that secures your validator keystore(s). You will need to re-enter this to decrypt them when
you setup your Ethereum validators.:
```

Generating the Seed Phrase (Mnemonic)

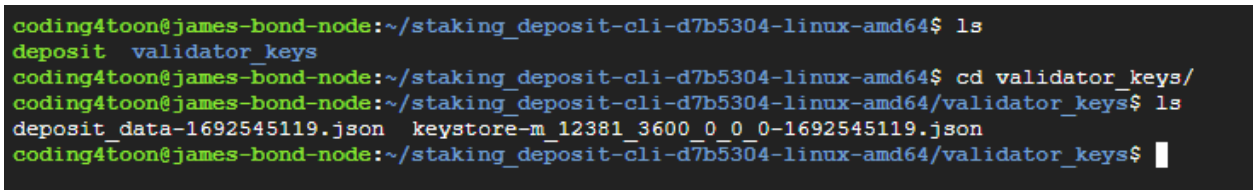
Post password setup, a seed phrase or mnemonic will be generated for you.

Agent T *- "This mnemonic is vital! Ensure you backup and store it in a secure location. In the future, it will be used to generate your withdrawal keys for the staked Goerli test network*

ETH or if you wish to add more validators. ***If, for any reason, you lose access to this mnemonic, you will not be able to retrieve your funds.****"*



Confirmation and Validator Key Creation
After verifying your mnemonic, the system will proceed to create your validator keys. The keys and deposit data file will be placed in the designated location. For instance: /home/coding4toon/staking_deposit-cli-d7b5304-linux-amd64/validator_keys



The file named deposit_data-[timestamp].json houses the public keys for the validators and information about the staking deposit. This file will be instrumental later in the guide during the Goerli test network ETH staking deposit process.

Files named keystore-[..].json store the encrypted validator signing keys. Each validator you fund will have its individual keystore file. These files will eventually be incorporated into the Consensus Client validator wallet for validation operations.

Agent T *- "If these files get lost or are accidentally deleted, fear not! They can be regenerated using the Staking Deposit Tool and your mnemonic with the existing-mnemonic command."*

Proceeding with the Lighthouse

Now that the deposit data, keystore files, validator password, and mnemonic have been generated and backed up, you're set to import the keystore files into Lighthouse.

Agent T *- "You'll effectively prepare your system for staking on the Goerli test network without any hassles! ….**Caution! Hold off on depositing any Goerli test network ETH at this point.** It's crucial to finish and validate your staking setup first. If the Goerli test network ETH deposits get activated and your staking system isn't prepared, you'll incur penalties due to inactivity, depleting your staked Goerli test network ETH balance."*

10.8- Importing Validator Keys

Setting Permissions for Validator Keystores

You'll start by ensuring the right permissions are set for the validator_keys directory, so replace <yourusername> with your actual username.

$ sudo chown -R coding4toon:coding4toon $HOME/staking_deposit-cli-d7b5304-linux-amd64/validator_keys

```
coding4toon@james-bond-node:~/staking_deposit-cli-d7b5304-linux-amd64/validator_keys$ sudo chown -R coding4toon
:coding4toon $HOME/staking_deposit-cli-d7b5304-linux-amd64/validator_keys
coding4toon@james-bond-node:~/staking_deposit-cli-d7b5304-linux-amd64/validator_keys$
```

Setting Up Lighthouse for Validator Keystores

Create a Directory

First, you'll create a directory to house the validator data. The logged-in user should have permission to access this directory since they'll be responsible for the import. Again, replace <yourusername> with your actual username.

$ sudo mkdir -p /var/lib/lighthouse

$ sudo chown -R coding4toon:coding4toon /var/lib/lighthouse

```
coding4toon@james-bond-node:~$ sudo mkdir -p /var/lib/lighthouse
coding4toon@james-bond-node:~$ sudo chown -R coding4toon:coding4toon /var/lib/lighthouse
coding4toon@james-bond-node:~$
```

Import Validator Keystores

Now, you'll carry out the validator keystore import. During this step, point the process to the location of the keystore-[..].json files.

$ /usr/local/bin/lighthouse --network goerli account validator import --directory /home/coding4toon/staking_deposit-cli-d7b5304-linux-amd64/validator_keys --datadir /var/lib/lighthouse

```
coding4toon@james-bond-node:~$ /usr/local/bin/lighthouse --network goerli account validator import --directory
/home/coding4toon/staking_deposit-cli-d7b5304-linux-amd64/validator_keys --datadir /var/lib/lighthouse
Running account manager for goerli network
validator-dir path: "/var/lib/lighthouse/validators"
WARNING: DO NOT USE THE ORIGINAL KEYSTORES TO VALIDATE WITH ANOTHER CLIENT, OR YOU WILL GET SLASHED.

Keystore found at "/home/coding4toon/staking_deposit-cli-d7b5304-linux-amd64/validator_keys/keystore-m_12381_36
00_0_0_0-1692545119.json":

 - Public key: 0xb04ceea4c9574bc7a9f33554479d8c55c6729d5f0ba77938251978bd626fc63220eac2bec03c5c50c95b5e2a51d8b2
05
 - UUID: a24967d3-1fa0-439f-8b26-7344eb7d1ab2

If you enter the password it will be stored as plain-text in validator_definitions.yml so that it is not requir
ed each time the validator client starts.

Enter the keystore password, or press enter to omit it:
```

Password Entry

The system will prompt you to input the password associated with the validator keys. This is the same password you designated when the keys were generated.

```
Enter the keystore password, or press enter to omit it:

Password is correct.

Successfully imported keystore.
Successfully updated validator_definitions.yml.

Successfully imported 1 validators (0 skipped).

WARNING: DO NOT USE THE ORIGINAL KEYSTORES TO VALIDATE WITH ANOTHER CLIENT, OR YOU WILL GET SLASHED.
coding4toon@james-bond-node:~$
```

Post import, the validator data will reside in /var/lib/lighthouse/validators.

Restoring System Permissions

To maintain the security and integrity of the system, restore the default permissions of the lighthouse directory after importing.

$ sudo chown -R root:root /var/lib/lighthouse

```
coding4toon@james-bond-node:~$ sudo chown -R root:root /var/lib/lighthouse
coding4toon@james-bond-node:~$
```

Agent T *- "At this point, the import process is concluded, and the wallet setup for your staking is in place."*

10.9- Configuring the Beacon Node Service

This ensures that the Lighthouse beacon node process will resume even after a system restart..

Service Account Setup
You'll be setting up an account exclusively for the service, making sure it's unable to login.
$ sudo useradd --no-create-home --shell /bin/false lighthousebeacon

```
coding4toon@james-bond-node:~$ sudo useradd --no-create-home --shell /bin/false lighthousebeacon
coding4toon@james-bond-node:~$
```

Data Directory Creation
Establish a directory to store the Lighthouse beacon node database.
$ sudo mkdir -p /var/lib/lighthouse/beacon

Set Permissions
Assign appropriate permissions to the directory.
$ sudo chown -R lighthousebeacon:lighthousebeacon /var/lib/lighthouse/beacon

```
coding4toon@james-bond-node:~$ sudo mkdir -p /var/lib/lighthouse/beacon
coding4toon@james-bond-node:~$ sudo chown -R lighthousebeacon:lighthousebeacon /var/lib/lighthouse/beacon
coding4toon@james-bond-node:~$
```

Creating a Config File
This file will be crucial for the systemd service setup.
$ sudo nano /etc/systemd/system/lighthousebeacon.service

Content for the Config File
Copy the given content into the file. This configures how the service operates.

```
[Unit]
Description=Lighthouse Consensus Client BN (Goerli Test Network)
Wants=network-online.target
After=network-online.target
[Service]
User=lighthousebeacon
Group=lighthousebeacon
Type=simple
Restart=always
RestartSec=5
ExecStart=/usr/local/bin/lighthouse bn \
 --network goerli \
 --datadir /var/lib/lighthouse \
 --http \
 --execution-endpoint http://localhost:8551 \
 --execution-jwt /var/lib/jwtsecret/jwt.hex \
```

--checkpoint-sync-url https://checkpoint-sync.goerli.ethpandaops.io \
--metrics
[Install]
WantedBy=multi-user.target

```
  GNU nano 4.8                    /etc/systemd/system/lighthousebeacon.service
[Unit]
Description=Lighthouse Consensus Client BN (Goerli Test Network)
Wants=network-online.target
After=network-online.target
[Service]
User=lighthousebeacon
Group=lighthousebeacon
Type=simple
Restart=always
RestartSec=5
ExecStart=/usr/local/bin/lighthouse bn \
 --network goerli \
 --datadir /var/lib/lighthouse \
 --http \
 --execution-endpoint http://localhost:8551 \
 --execution-jwt /var/lib/jwtsecret/jwt.hex \
 --checkpoint-sync-url https://checkpoint-sync.goerli.ethpandaops.io \
 --metrics
[Install]
WantedBy=multi-user.target
```

Agent T *- "Be sure to set a valid checkpoint sync URL. See below for more information. For example: https://checkpoint-sync.goerli.ethpandaops.io"*

- bn subcommand instructs the lighthouse binary to run as a beacon node.
- --http Exposes an http endpoint which is used by the validator client to connect to the beacon node.
- --execution-endpoint http://127.0.0.1:8551 The address of the Execution Client. Should be the same for all Execution Clients detailed in this guide.
- --execution-jwt /var/lib/jwtsecret/jwt.hex The path to the JWT file that is required for authenticated communication between the Execution and Consensus clients.
- --checkpoint-sync-url Enables the Checkpoint Sync feature to greatly speed up the Beacon Chain Node sync.

Agent T *- "Provide a URL to a synced Beacon Chain Node for the sync. Make sure you choose a Goerli endpoint."*

Ressources

https://eth-clients.github.io/checkpoint-sync-endpoints/

Goerli

Name	State	Verification	Block	Endpoint	Contact details	Notes
Lodestar (ChainSafe)	✅	✅	❌	https://beaconstate-goerli.chainsafe.io	@lodestar_eth	
beaconcha.in	✅	✅	❌	https://sync-goerli.beaconcha.in	@beaconcha_in	
BeaconState.info	✅	✅	❌	https://goerli.beaconstate.info		
Stakely	✅	✅	❌	https://prater-checkpoint-sync.stakely.io	@Stakely_io	
EthStaker	✅	✅	❌	https://goerli.beaconstate.ethstaker.cc/	@remy_roy	
invis.tools	✅	✅	❌	https://goerli-sync.invis.tools	@0xinvis	
EF DevOps	✅	✅	❌	https://checkpoint-sync.goerli.ethpandaops.io	@samcmau @savid	
Sigma Prime	✅	✅	❌	https://prater.checkpoint.sigp.io		
Nimbus	✅	❌	✅	http://unstable.prater.beacon-api.nimbus.team/	@ethnimbus	

Reload and Start

Make sure systemd is up to date with the changes, then start the service.

```
$ sudo systemctl daemon-reload
$ sudo systemctl start lighthousebeacon
```

Status Check

Validate if the service is operational.

```
$ sudo systemctl status lighthousebeacon
```

```
coding4toon@james-bond-node:~$ sudo systemctl daemon-reload
coding4toon@james-bond-node:~$ sudo systemctl start lighthousebeacon
coding4toon@james-bond-node:~$ sudo systemctl status lighthousebeacon
● lighthousebeacon.service - Lighthouse Consensus Client BN (Goerli Test Network)
     Loaded: loaded (/etc/systemd/system/lighthousebeacon.service; disabled; vendor preset: enabled)
     Active: active (running) since Sun 2023-08-20 16:04:13 UTC; 9s ago
   Main PID: 8246 (lighthouse)
      Tasks: 25 (limit: 38496)
     Memory: 546.1M
     CGroup: /system.slice/lighthousebeacon.service
             └─8246 /usr/local/bin/lighthouse bn --network goerli --datadir /var/lib/lighthouse --http --execu>

Aug 20 16:04:13 james-bond-node lighthouse[8246]: Aug 20 16:04:13.485 INFO Lighthouse started                >
Aug 20 16:04:13 james-bond-node lighthouse[8246]: Aug 20 16:04:13.485 INFO Configured for network           >
Aug 20 16:04:13 james-bond-node lighthouse[8246]: Aug 20 16:04:13.485 INFO Data directory initialised       >
Aug 20 16:04:13 james-bond-node lighthouse[8246]: Aug 20 16:04:13.485 INFO Deposit contract                 >
Aug 20 16:04:13 james-bond-node lighthouse[8246]: Aug 20 16:04:13.571 INFO Starting checkpoint sync         >
Aug 20 16:04:13 james-bond-node lighthouse[8246]: Aug 20 16:04:13.973 WARN Remote BN does not support EIP-4881>
Aug 20 16:04:17 james-bond-node lighthouse[8246]: Aug 20 16:04:17.326 INFO Loaded checkpoint block and state  >
Aug 20 16:04:21 james-bond-node lighthouse[8246]: Aug 20 16:04:21.083 INFO Block production enabled          >
Aug 20 16:04:21 james-bond-node lighthouse[8246]: Aug 20 16:04:21.332 WARN Execution endpoint is not synced >
Aug 20 16:04:21 james-bond-node lighthouse[8246]: Aug 20 16:04:21.332 ERRO Error updating deposit contract cac>
lines 1-19/19 (END)
```

You should see "active (running)" in green.

To exit press Q.

Agent T *- "The sync will begin immediately.In order to be able to stake both the Execution Client and the Consensus Client must be fully synced."*

Viewing the Sync

To observe the synchronization process or to check for issues, utilize the journal output.

$ sudo journalctl -fu lighthousebeacon

```
coding4toon@james-bond-node:~$ sudo journalctl -fu lighthousebeacon
-- Logs begin at Sun 2023-08-20 12:55:08 UTC. --
Aug 20 16:04:56 james-bond-node lighthouse[8246]: Aug 20 16:04:56.966 INFO Deterministic long lived subnets ena
bled, subscription_duration_in_epochs: 256, subnets_per_node: 2, service: attestation_service
Aug 20 16:04:56 james-bond-node lighthouse[8246]: Aug 20 16:04:56.966 INFO Subscribing to long-lived subnets
    subnets: [SubnetId(12), SubnetId(13)], service: attestation_service
Aug 20 16:04:56 james-bond-node lighthouse[8246]: Aug 20 16:04:56.967 INFO HTTP API started
    listen_address: 127.0.0.1:5052
Aug 20 16:04:56 james-bond-node lighthouse[8246]: Aug 20 16:04:56.968 INFO Metrics HTTP server started
    listen_address: 127.0.0.1:5054
Aug 20 16:04:56 james-bond-node lighthouse[8246]: Aug 20 16:04:56.977 INFO Execution engine online
    service: exec
Aug 20 16:04:56 james-bond-node lighthouse[8246]: Aug 20 16:04:56.977 INFO Issuing forkchoiceUpdated
    forkchoice_state: ForkchoiceState { head_block_hash: 0xeac2f2a095fe2b7e864577e1d35553d1b2e694e6dabab3fa4a7e
8a171efbd671, safe_block_hash: 0xeac2f2a095fe2b7e864577e1d35553d1b2e694e6dabab3fa4a7e8a171efbd671, finalized_bl
ock_hash: 0xeac2f2a095fe2b7e864577e1d35553d1b2e694e6dabab3fa4a7e8a171efbd671 }, service: exec
Aug 20 16:04:57 james-bond-node lighthouse[8246]: Aug 20 16:04:57.317 INFO Execution payloads are pruned
    service: freezer_db
Aug 20 16:04:57 james-bond-node lighthouse[8246]: Aug 20 16:04:57.709 INFO Sync state updated
    new_state: Syncing Finalized Chain, old_state: Stalled, service: sync
Aug 20 16:05:06 james-bond-node lighthouse[8246]: Aug 20 16:05:06.001 INFO Syncing
    est_time: --, distance: 6961 slots (23 hrs 12 mins), peers: 3, service: slot_notifier
Aug 20 16:05:06 james-bond-node lighthouse[8246]: Aug 20 16:05:06.002 WARN Syncing deposit contract block cache
    est_blocks_remaining: initializing deposits, service: slot_notifier
Aug 20 16:05:07 james-bond-node lighthouse[8246]: Aug 20 16:05:07.177 INFO UPnP not available
    error: IO error: Resource temporarily unavailable (os error 11), service: UPnP
Aug 20 16:05:18 james-bond-node lighthouse[8246]: Aug 20 16:05:18.001 INFO Syncing
    est_time: 21 mins, speed: 5.33 slots/sec, distance: 6898 slots (22 hrs 59 mins), peers: 6, service: slot_no
tifier
Aug 20 16:05:18 james-bond-node lighthouse[8246]: Aug 20 16:05:18.001 WARN Syncing deposit contract block cache
```

Agent T *- "*The Lighthouse client will not attempt to perform validator duties without a connection to a fully synced Execution Client."

Agent T *- "Me again,* since Besu (Execution client) was waiting for the consensus client, it'll now begin synchronization."

$ sudo journalctl -fu besu

```
coding4toon@james-bond-node:~$ sudo journalctl -fu besu
-- Logs begin at Sun 2023-08-20 12:55:08 UTC. --
Aug 20 16:03:48 james-bond-node besu[7756]: 2023-08-20 16:03:48.360+00:00 | EthScheduler-Timer-0 | INFO  | Pivo
tSelectorFromSafeBlock | Waiting for consensus client, this may be because your consensus client is still synci
ng
Aug 20 16:03:51 james-bond-node besu[7756]: 2023-08-20 16:03:51.122+00:00 | vert.x-eventloop-thread-0 | WARN  |
 EngineQosTimer | Execution engine not called in 120 seconds, consensus client may not be connected
Aug 20 16:04:06 james-bond-node systemd[1]: /etc/systemd/system/besu.service:1: Assignment outside of section.
Ignoring.
Aug 20 16:04:48 james-bond-node besu[7756]: 2023-08-20 16:04:48.363+00:00 | EthScheduler-Timer-0 | INFO  | Pivo
tSelectorFromSafeBlock | Waiting for consensus client, this may be because your consensus client is still synci
ng
Aug 20 16:04:58 james-bond-node besu[7756]: 2023-08-20 16:04:58.407+00:00 | nioEventLoopGroup-3-7 | INFO  | Sna
pWorldStateDownloader | Downloading world state from peers for pivot block 9545858 (0xeac2f2a095fe2b7e864577e1d
35553d1b2e694e6dabab3fa4a7e8a171efbd671). State root 0x0c709ec36a21c4f938553be5463d32c303182b5dd6d60f841d709398
fd729620 pending requests 0
Aug 20 16:04:58 james-bond-node besu[7756]: 2023-08-20 16:04:58.536+00:00 | nioEventLoopGroup-3-7 | INFO  | Bon
saiWorldStateKeyValueStorage | Bonsai flat db mode found PARTIAL
Aug 20 16:06:18 james-bond-node systemd[1]: /etc/systemd/system/besu.service:1: Assignment outside of section.
Ignoring.
Aug 20 16:07:09 james-bond-node besu[7756]: 2023-08-20 16:07:09.659+00:00 | EthScheduler-Services-42 (importBlo
ck) | INFO  | FastImportBlocksStep | Block import progress: 99600 of 9545858 (1%)
Aug 20 16:08:08 james-bond-node besu[7756]: 2023-08-20 16:08:08.793+00:00 | EthScheduler-Services-42 (importBlo
ck) | INFO  | FastImportBlocksStep | Block import progress: 197400 of 9545858 (2%)
Aug 20 16:09:09 james-bond-node besu[7756]: 2023-08-20 16:09:09.971+00:00 | EthScheduler-Services-42 (importBlo
ck) | INFO  | FastImportBlocksStep | Block import progress: 291000 of 9545858 (3%)
```

Ensure Service Autostart

This step confirms that the service will always kickstart post reboot.

$ sudo systemctl enable lighthousebeacon

```
coding4toon@james-bond-node:~$ sudo systemctl enable lighthousebeacon
Created symlink /etc/systemd/system/multi-user.target.wants/lighthousebeacon.service → /etc/systemd/system/ligh
thousebeacon.service.
coding4toon@james-bond-node:~$
```

Agent T *- "This process ensures the Lighthouse beacon node runs efficiently as a service. If it isn't functioning correctly, it's advised to revisit the steps. Always make sure both Execution Client and Consensus Client are fully synchronized before initiating staking operations."*

10.10- Configuring the Validator Service

Establish Validator Node Account

You need an account under which the validator node will operate. This account won't have login access to the server.

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

```
coding4toon@james-bond-node:~$ sudo useradd --no-create-home --shell /bin/false lighthousevalidator
coding4toon@james-bond-node:~$
```

Directory Permissions for the Validator Node

After the validator import process, the /var/lib/lighthouse/validators directory was created. Set its permissions to allow the lighthousevalidator account to make changes.

$ sudo chown -R lighthousevalidator:lighthousevalidator /var/lib/lighthouse/validators

```
coding4toon@james-bond-node:~$ sudo chown -R lighthousevalidator:lighthousevalidator /var/lib/light
house/validators
coding4toon@james-bond-node:~$
```

Configuring the Service

Craft a systemd service file to save the service's configuration.

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

Embed the provided settings into the file. This dictates how the Lighthouse Validator Node will operate.

```
[Unit]
Description=Lighthouse Consensus Client VC (Goerli Test Network)
Wants=network-online.target
After=network-online.target
[Service]
User=lighthousevalidator
Group=lighthousevalidator
Type=simple
Restart=always
RestartSec=5
ExecStart=/usr/local/bin/lighthouse vc \
 --network goerli \
 --datadir /var/lib/lighthouse \
 --suggested-fee-recipient 0x26Eb144aC5C852999705eE73633b42BB922C2853 \
 --graffiti "im-bond-james-bond"
[Install]
WantedBy=multi-user.target
```

```
  GNU nano 4.8              /etc/systemd/system/lighthousevalidator.service                Modified
[Unit]
Description=Lighthouse Consensus Client VC (Goerli Test Network)
Wants=network-online.target
After=network-online.target
[Service]
User=lighthousevalidator
Group=lighthousevalidator
Type=simple
Restart=always
RestartSec=5
ExecStart=/usr/local/bin/lighthouse vc \
 --network goerli \
 --datadir /var/lib/lighthouse \
 --suggested-fee-recipient 0x26Eb144aC5C852999705eE73633b42BB922C2853 \
 --graffiti "im-bond-james-bond"
[Install]
WantedBy=multi-user.target
```

FeeRecipientAddress

Ensure you pinpoint the fee recipient address to a valid Ethereum address you control to amass the validator fees. Example: 0x26Eb144aC5C852999705eE73633b42BB922C2853

Key Flags Elucidated

- vc: Commands the Lighthouse binary to function as a validator node.
- --suggested-fee-recipient: Validators are eligible for tips from user transactions. Input a legitimate Ethereum address you possess to designate the recipient of these tips.
- --graffiti: Allow for a customizable graffiti string on the beacon block. Use a general graffiti string and dodge using identifiers. Example:--graffiti="im-bond-james-bond".

Reload and Start Service

Refresh systemd to update changes and initiate the lighthouse validator service.

```
$ sudo systemctl daemon-reload
$ sudo systemctl start lighthousevalidator
```

Check Service Status

Ensure the service is running as expected.

```
$ sudo systemctl status lighthousevalidator
```

```
coding4toon@james-bond-node:~$ sudo systemctl daemon-reload
coding4toon@james-bond-node:~$ sudo systemctl start lighthousevalidator
coding4toon@james-bond-node:~$ sudo systemctl status lighthousevalidator
● lighthousevalidator.service - Lighthouse Consensus Client VC (Goerli Test Network)
     Loaded: loaded (/etc/systemd/system/lighthousevalidator.service; disabled; vendor preset: ena>
     Active: active (running) since Sun 2023-08-20 19:31:19 UTC; 8s ago
   Main PID: 8855 (lighthouse)
      Tasks: 19 (limit: 38496)
     Memory: 51.5M
     CGroup: /system.slice/lighthousevalidator.service
             └─8855 /usr/local/bin/lighthouse vc --network goerli --datadir /var/lib/lighthouse -->

Aug 20 19:31:21 james-bond-node lighthouse[8855]: Aug 20 19:31:21.027 INFO Pruning slashing protec>
Aug 20 19:31:21 james-bond-node lighthouse[8855]: Aug 20 19:31:21.027 INFO Completed pruning of sl>
Aug 20 19:31:21 james-bond-node lighthouse[8855]: Aug 20 19:31:21.029 INFO Genesis has already occ>
Aug 20 19:31:21 james-bond-node lighthouse[8855]: Aug 20 19:31:21.029 INFO Block production servic>
Aug 20 19:31:21 james-bond-node lighthouse[8855]: Aug 20 19:31:21.030 INFO Attestation production >
Aug 20 19:31:21 james-bond-node lighthouse[8855]: Aug 20 19:31:21.030 INFO Sync committee service >
Aug 20 19:31:21 james-bond-node lighthouse[8855]: Aug 20 19:31:21.030 INFO Validator registration >
Aug 20 19:31:21 james-bond-node lighthouse[8855]: Aug 20 19:31:21.030 INFO Proposer preparation se>
Aug 20 19:31:21 james-bond-node lighthouse[8855]: Aug 20 19:31:21.030 INFO Doppelganger protection>
Aug 20 19:31:21 james-bond-node lighthouse[8855]: Aug 20 19:31:21.030 INFO HTTP API server is disa>
lines 1-19/19 (END)
```

If you see "active (running)" in green, it's working. If not, review and redo the previous steps. To exit, press Q.

Agent T *- "Your system will start synchronizing. Remember: For staking, both the Execution Client and Consensus Client need to be fully synchronized."*

Monitor Progress

To track syncing progress or spot errors, use:

$ sudo journalctl -fu lighthousevalidator

```
coding4toon@james-bond-node:~$ sudo journalctl -fu lighthousevalidator
-- Logs begin at Sun 2023-08-20 12:55:08 UTC. --
Aug 20 19:31:30 james-bond-node lighthouse[8855]: Aug 20 19:31:30.001 INFO Connected to beacon node
(s)             synced: 1, available: 1, total: 1, service: notifier
Aug 20 19:31:30 james-bond-node lighthouse[8855]: Aug 20 19:31:30.001 INFO Awaiting activation
                slot: 6337657, epoch: 198051, validators: 1, service: notifier
Aug 20 19:31:42 james-bond-node lighthouse[8855]: Aug 20 19:31:42.001 INFO Connected to beacon node
(s)             synced: 1, available: 1, total: 1, service: notifier
Aug 20 19:31:42 james-bond-node lighthouse[8855]: Aug 20 19:31:42.001 INFO Awaiting activation
                slot: 6337658, epoch: 198051, validators: 1, service: notifier
Aug 20 19:31:54 james-bond-node lighthouse[8855]: Aug 20 19:31:54.001 INFO Connected to beacon node
(s)             synced: 1, available: 1, total: 1, service: notifier
Aug 20 19:31:54 james-bond-node lighthouse[8855]: Aug 20 19:31:54.001 INFO Awaiting activation
                slot: 6337659, epoch: 198051, validators: 1, service: notifier
Aug 20 19:32:06 james-bond-node lighthouse[8855]: Aug 20 19:32:06.001 INFO Connected to beacon node
(s)             synced: 1, available: 1, total: 1, service: notifier
Aug 20 19:32:06 james-bond-node lighthouse[8855]: Aug 20 19:32:06.001 INFO Awaiting activation
                slot: 6337660, epoch: 198051, validators: 1, service: notifier
Aug 20 19:32:18 james-bond-node lighthouse[8855]: Aug 20 19:32:18.001 INFO Connected to beacon node
(s)             synced: 1, available: 1, total: 1, service: notifier
Aug 20 19:32:18 james-bond-node lighthouse[8855]: Aug 20 19:32:18.002 INFO Awaiting activation
                slot: 6337661, epoch: 198051, validators: 1, service: notifier
Aug 20 19:32:30 james-bond-node lighthouse[8855]: Aug 20 19:32:30.001 INFO Connected to beacon node
(s)             synced: 1, available: 1, total: 1, service: notifier
Aug 20 19:32:30 james-bond-node lighthouse[8855]: Aug 20 19:32:30.002 INFO Awaiting activation
                slot: 6337662, epoch: 198051, validators: 1, service: notifier
```

Enable Auto-Start

Set the validator service to automatically launch after system reboots.

$ sudo systemctl enable lighthousevalidator

```
coding4toon@james-bond-node:~$ sudo systemctl enable lighthousevalidator
Created symlink /etc/systemd/system/multi-user.target.wants/lighthousevalidator.service → /etc/syst
emd/system/lighthousevalidator.service.
coding4toon@james-bond-node:~$
```

 ***Agent T*** *- "**Congratulations!** You've successfully set up and initiated the Lighthouse Consensus Client. Up next, you'll deposit to activate your validators on the network."*

10.11- From “in Progress” to “Fully Synced” Client

Both clients (Bezu and Lighthouse) need to "sync" or download the most recent state of the blockchain to work correctly.

Checking If They Are Fully Synced

For both Bezu and Lighthouse, when they are syncing, you will see logs that indicate they are downloading blocks or catching up with the latest state of the blockchain.

The following are example statuses from in progress client journal log output.

$ sudo journalctl -fu lighthousebeacon

```
coding4toon@james-bond-node:~$ sudo journalctl -fu lighthousebeacon
-- Logs begin at Sun 2023-08-20 12:55:08 UTC. --
Aug 20 19:37:18 james-bond-node lighthouse[8246]: Aug 20 19:37:18.002 INFO Downloading historical b
locks           est_time: 15 hrs 4 mins, speed: 15.98 slots/sec, distance: 867171 slots (17 weeks 1
 days), service: slot_notifier
Aug 20 19:37:18 james-bond-node lighthouse[8246]: Aug 20 19:37:18.002 WARN Head is optimistic
                execution_block_hash: 0xdc9b1192fa531ed39814f181f54774a845e66ce9a4bb5a9b9443e8dadd5
734b2, info: chain not fully verified, block and attestation production disabled until execution en
gine syncs, service: slot_notifier
Aug 20 19:37:18 james-bond-node lighthouse[8246]: Aug 20 19:37:18.002 INFO Synced
                slot: 6337686, block:    …  empty, epoch: 198052, finalized_epoch: 198050, finalize
d_root: 0x82d9…48b0, exec_hash: 0xdc9b…34b2 (unverified), peers: 77, service: slot_notifier
Aug 20 19:37:18 james-bond-node lighthouse[8246]: Aug 20 19:37:18.002 WARN Syncing deposit contract
 block cache    est_blocks_remaining: initializing deposits, service: slot_notifier
Aug 20 19:37:21 james-bond-node lighthouse[8246]: Aug 20 19:37:21.092 WARN Execution endpoint is no
t synced        last_seen_block_unix_timestamp: 1548854791, endpoint: http://localhost:8551/, auth=
true, service: deposit_contract_rpc
Aug 20 19:37:21 james-bond-node lighthouse[8246]: Aug 20 19:37:21.092 ERRO Error updating deposit c
ontract cache   error: Failed to get remote head and new block ranges: EndpointError(FarBehind), re
try_millis: 60000, service: deposit_contract_rpc
Aug 20 19:37:25 james-bond-node lighthouse[8246]: Aug 20 19:37:25.208 INFO New block received
                root: 0x04c4975d6be710479e51252f7ad7e4177218f440c5aa44c9409c9a63aafddafe, slot: 633
7687
Aug 20 19:37:30 james-bond-node lighthouse[8246]: Aug 20 19:37:30.001 WARN Head is optimistic
                execution_block_hash: 0x3b66fa49958f88fc99dcae3b398247c2808bc3d6874b4c8f3c78de8b7a6
4d472, info: chain not fully verified, block and attestation production disabled until execution en
gine syncs, service: slot_notifier
Aug 20 19:37:30 james-bond-node lighthouse[8246]: Aug 20 19:37:30.001 INFO Synced
                slot: 6337687, block: 0x04c4…dafe, epoch: 198052, finalized_epoch: 198050, finalize
d_root: 0x82d9…48b0, exec_hash: 0x3b66…d472 (unverified), peers: 77, service: slot_notifier
Aug 20 19:37:30 james-bond-node lighthouse[8246]: Aug 20 19:37:30.001 WARN Syncing deposit contract
 block cache    est_blocks_remaining: initializing deposits, service: slot_notifier
Aug 20 19:37:37 james-bond-node lighthouse[8246]: Aug 20 19:37:37.315 INFO New block received
                root: 0x609f8bdf5629c78f05b274ceb8606bd491e6cb01c2d00512ce97626991844721, slot: 633
7688
Aug 20 19:37:42 james-bond-node lighthouse[8246]: Aug 20 19:37:42.002 WARN Head is optimistic
                execution_block_hash: 0xeacd2bc86dde58a7e8cbaf614d45d84dc72baa16ed42070bba91b17716b
26f92, info: chain not fully verified, block and attestation production disabled until execution en
gine syncs, service: slot_notifier
```

$ sudo journalctl -fu lighthousevalidator

```
coding4toon@james-bond-node:~$ sudo journalctl -fu lighthousevalidator
-- Logs begin at Sun 2023-08-20 12:55:08 UTC. --
Aug 20 19:38:30 james-bond-node lighthouse[8855]: Aug 20 19:38:30.001 INFO Connected to beacon node
(s)             synced: 1, available: 1, total: 1, service: notifier
Aug 20 19:38:30 james-bond-node lighthouse[8855]: Aug 20 19:38:30.001 INFO Awaiting activation
                slot: 6337692, epoch: 198052, validators: 1, service: notifier
Aug 20 19:38:42 james-bond-node lighthouse[8855]: Aug 20 19:38:42.001 INFO Connected to beacon node
(s)             synced: 1, available: 1, total: 1, service: notifier
Aug 20 19:38:42 james-bond-node lighthouse[8855]: Aug 20 19:38:42.001 INFO Awaiting activation
                slot: 6337693, epoch: 198052, validators: 1, service: notifier
Aug 20 19:38:54 james-bond-node lighthouse[8855]: Aug 20 19:38:54.001 INFO Connected to beacon node
(s)             synced: 1, available: 1, total: 1, service: notifier
Aug 20 19:38:54 james-bond-node lighthouse[8855]: Aug 20 19:38:54.001 INFO Awaiting activation
                slot: 6337694, epoch: 198052, validators: 1, service: notifier
Aug 20 19:39:06 james-bond-node lighthouse[8855]: Aug 20 19:39:06.001 INFO Connected to beacon node
(s)             synced: 1, available: 1, total: 1, service: notifier
Aug 20 19:39:06 james-bond-node lighthouse[8855]: Aug 20 19:39:06.001 INFO Awaiting activation
                slot: 6337695, epoch: 198052, validators: 1, service: notifier
Aug 20 19:39:18 james-bond-node lighthouse[8855]: Aug 20 19:39:18.001 INFO Connected to beacon node
(s)             synced: 1, available: 1, total: 1, service: notifier
Aug 20 19:39:18 james-bond-node lighthouse[8855]: Aug 20 19:39:18.001 INFO Awaiting activation
                slot: 6337696, epoch: 198053, validators: 1, service: notifier
```

$ sudo journalctl -fu besu

```
coding4toon@james-bond-node:~$ sudo journalctl -fu besu
-- Logs begin at Sun 2023-08-20 12:55:08 UTC. --
Aug 20 19:35:23 james-bond-node besu[7756]: 2023-08-20 19:35:23.068+00:00 | EthScheduler-Services-4
2 (importBlock) | INFO  | FastImportBlocksStep | Block import progress: 7865000 of 9545858 (82%)
Aug 20 19:35:53 james-bond-node besu[7756]: 2023-08-20 19:35:53.887+00:00 | EthScheduler-Services-4
2 (importBlock) | INFO  | FastImportBlocksStep | Block import progress: 7871800 of 9545858 (82%)
Aug 20 19:36:24 james-bond-node besu[7756]: 2023-08-20 19:36:24.044+00:00 | EthScheduler-Services-4
2 (importBlock) | INFO  | FastImportBlocksStep | Block import progress: 7878800 of 9545858 (82%)
Aug 20 19:36:54 james-bond-node besu[7756]: 2023-08-20 19:36:54.133+00:00 | EthScheduler-Services-4
2 (importBlock) | INFO  | FastImportBlocksStep | Block import progress: 7885600 of 9545858 (82%)
Aug 20 19:37:25 james-bond-node besu[7756]: 2023-08-20 19:37:25.048+00:00 | EthScheduler-Services-4
2 (importBlock) | INFO  | FastImportBlocksStep | Block import progress: 7892400 of 9545858 (82%)
Aug 20 19:37:55 james-bond-node besu[7756]: 2023-08-20 19:37:55.937+00:00 | EthScheduler-Services-4
2 (importBlock) | INFO  | FastImportBlocksStep | Block import progress: 7900200 of 9545858 (82%)
Aug 20 19:38:26 james-bond-node besu[7756]: 2023-08-20 19:38:26.133+00:00 | EthScheduler-Services-4
2 (importBlock) | INFO  | FastImportBlocksStep | Block import progress: 7907600 of 9545858 (82%)
Aug 20 19:38:56 james-bond-node besu[7756]: 2023-08-20 19:38:56.717+00:00 | EthScheduler-Services-4
2 (importBlock) | INFO  | FastImportBlocksStep | Block import progress: 7913600 of 9545858 (82%)
Aug 20 19:39:26 james-bond-node besu[7756]: 2023-08-20 19:39:26.777+00:00 | EthScheduler-Services-4
2 (importBlock) | INFO  | FastImportBlocksStep | Block import progress: 7919600 of 9545858 (82%)
Aug 20 19:39:57 james-bond-node besu[7756]: 2023-08-20 19:39:57.825+00:00 | EthScheduler-Services-4
2 (importBlock) | INFO  | FastImportBlocksStep | Block import progress: 7927000 of 9545858 (83%)
```

Agent *T* - *"When they are fully synced, these messages will either slow down significantly or you will see messages indicating that they are now in sync with the current state."*

How Much Time is Needed

The time required for these clients to sync can vary greatly based on several factors:

- **Internet Speed**

A faster connection will download blocks more quickly.

- **Computer Hardware**

More powerful computers can process blocks faster.

- **Blockchain Size**

The bigger the blockchain, the longer it takes.

- **Network Conditions**

If many nodes are online and responsive, syncing can be faster.

 **Agent *T*** - *“For a new setup, initial synchronization can take several hours to a few days. Once fully synced, keeping up-to-date usually takes much less time as you're just processing new blocks.”*

10.12- Funding the Validator

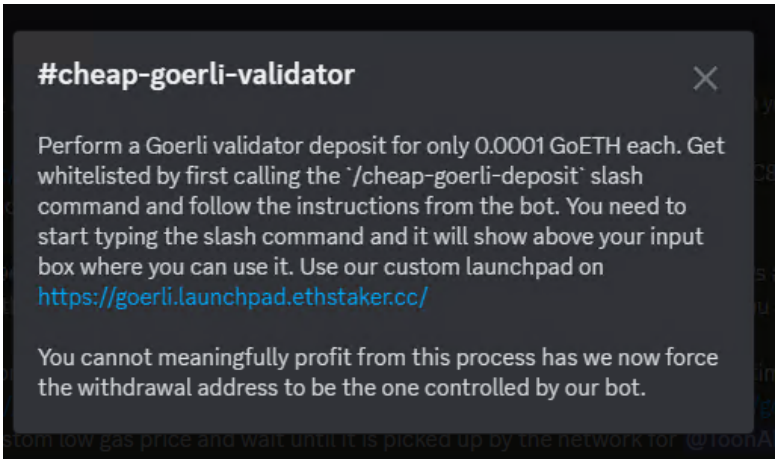
Now that the Consensus Client is up and running, to actually begin staking on the Goerli test network you will need to deposit Goerli test network ETH to fund your validator.

Getting Goerli Test Network Money (ETH)

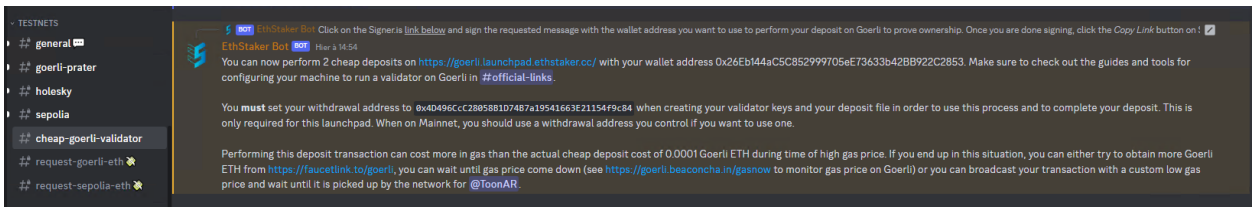
- Each validator needs 32 Goerli test ETH, plus a little extra for transaction costs.
- To get this ETH, we'll use a helpful **bot on the EthStaker Discord**.
- Join the **EthStaker Discord** and find the **#cheap-goerli-validator channel**. There, you'll see easy steps to use the bot and to be whitelisted.

Ressources

https://discord.com/invite/ucsTcA2wTq



Agent *T* - *"Why? We want real people, not spammy computer programs, to use the system. To prove you're real, you'll "sign" using your digital wallet."*

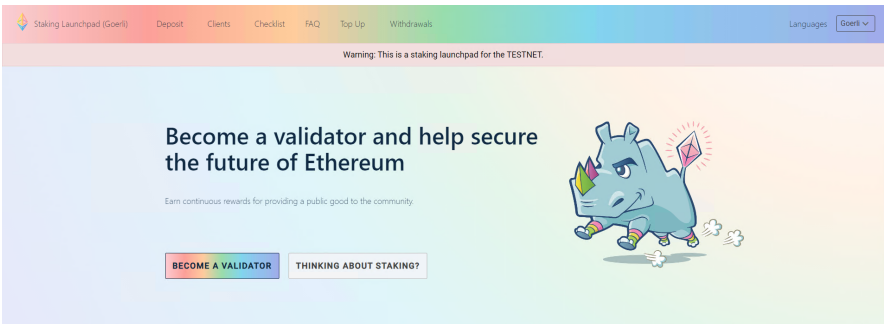


Making the Deposit

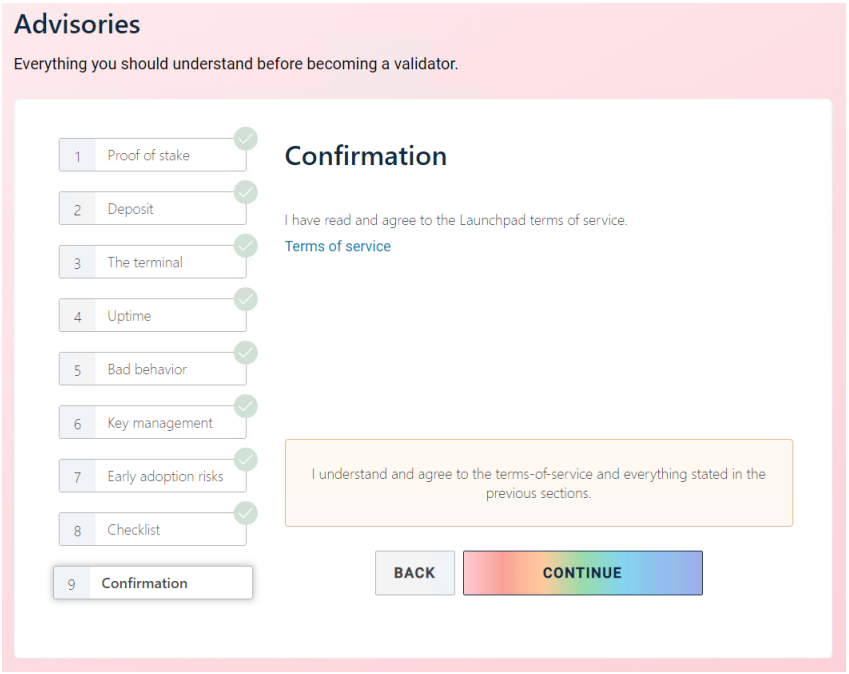
Once you have been whitelisted, you'll deposit GoETH into a special account on the Goerli staking website.

Agent *T* - *"But, wait! Ensure your Execution Client and Consensus Client are updated. If they're not and you start staking, you could lose some money because of inactivity fees."*

Visit the staking website: https://goerli.launchpad.ethstaker.cc/en/



Click "Become a Validator". Read the warnings, then keep clicking until you see "Generate Key Pairs".

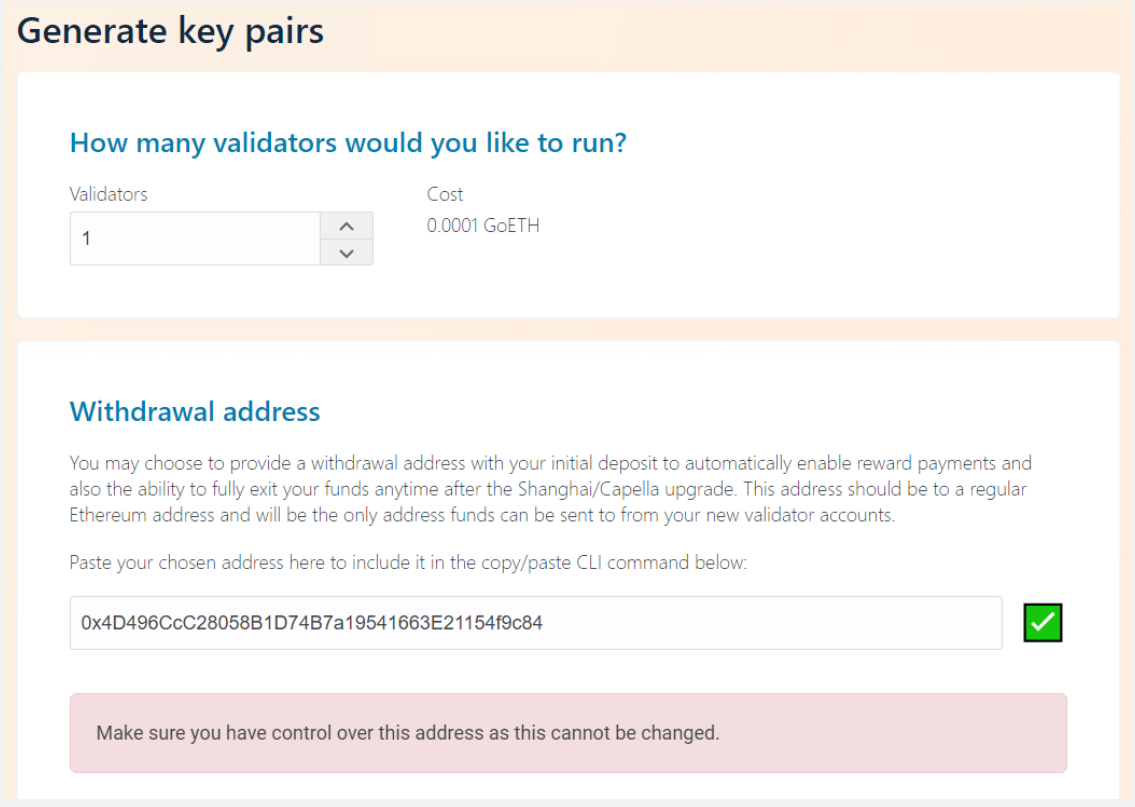


Pick Your Validator Number
You'll need to decide on how many validators you want. For our current setup, you should choose one validator.

Set Up Your Withdrawal Address
Use the bot's address as your withdrawal destination. This helps you get a discount!

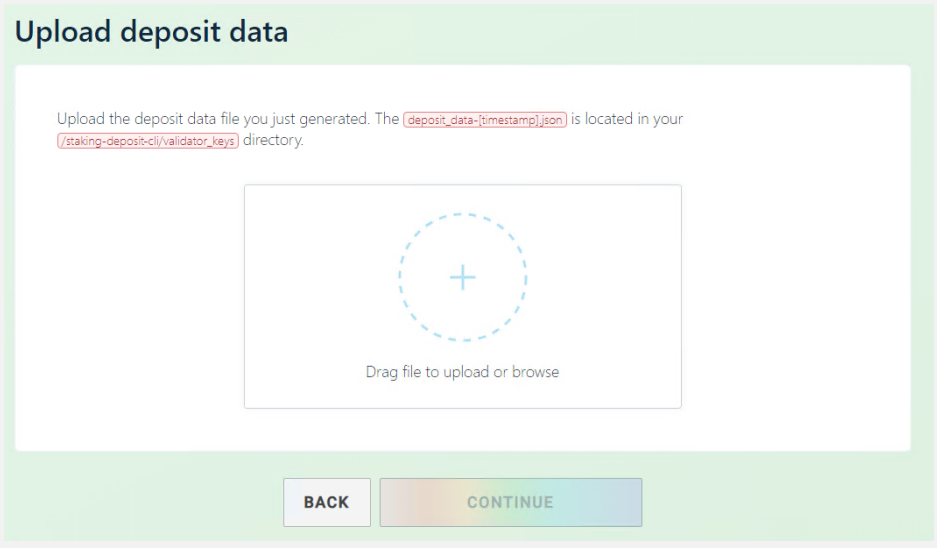
Cost Breakdown
Without the bot: The cost is 32 GoETH.
With the bot: The price drops dramatically to just 0.0001 GoETH.

By using the bot's address, you'll save a significant amount on your deposit in order to set up and run your node on testnet.



Agent *T* - *"This is like using a coupon code when shopping online – it helps reduce your costs!"*

Prepare Your Deposit Data
You'll first need to get your deposit data ready. Think of this as a special file the website needs from you.

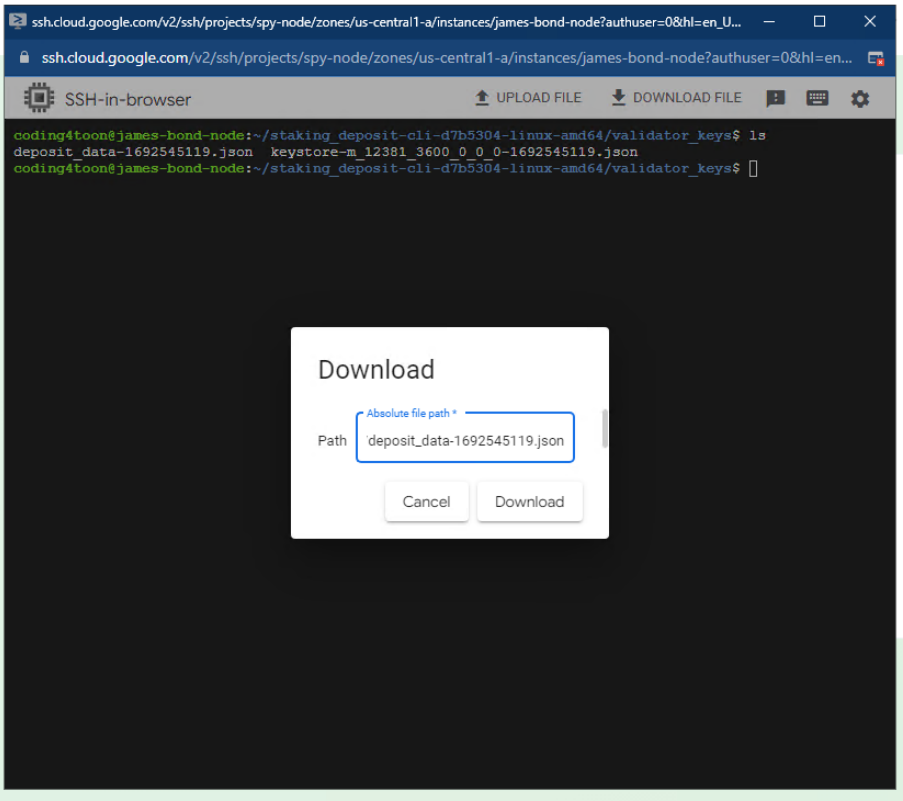


Using Google Cloud Platform (GCP)
GCP makes it super easy to get this file from a virtual computer, like "james-bond-node."

Locate where your deposit data is stored on the VM.
For this example, it's in:
/home/coding4toon/staking_deposit-cli-d7b5304-linux-amd64/validator_keys/deposit_data-1692545119.json

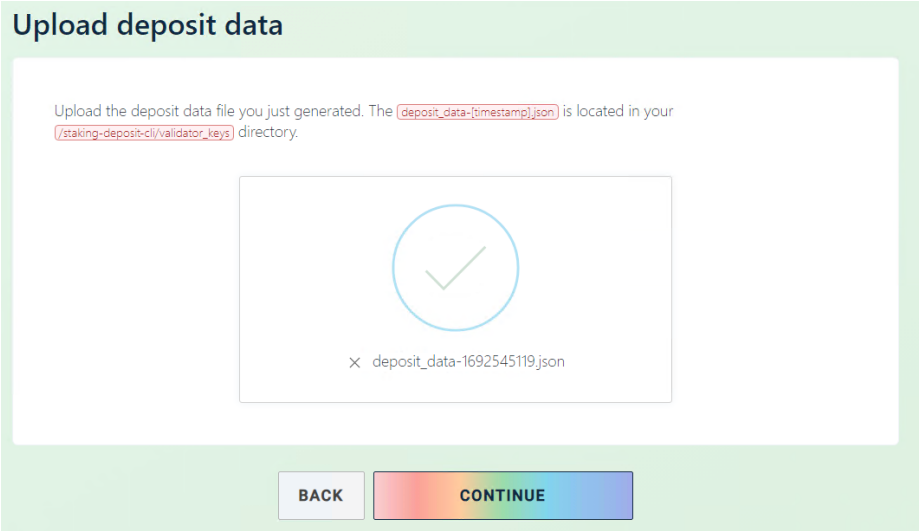
```
coding4toon@james-bond-node:~$ cd staking_deposit-cli-d7b5304-linux-amd64/
coding4toon@james-bond-node:~/staking_deposit-cli-d7b5304-linux-amd64$ cd validator_keys/
coding4toon@james-bond-node:~/staking_deposit-cli-d7b5304-linux-amd64/validator_keys$ ls
deposit_data-1692545119.json  keystore-m_12381_3600_0_0_0-1692545119.json
coding4toon@james-bond-node:~/staking_deposit-cli-d7b5304-linux-amd64/validator_keys$
```

Now, use the "download" feature in GCP to move this data from the virtual machine to your personal computer.



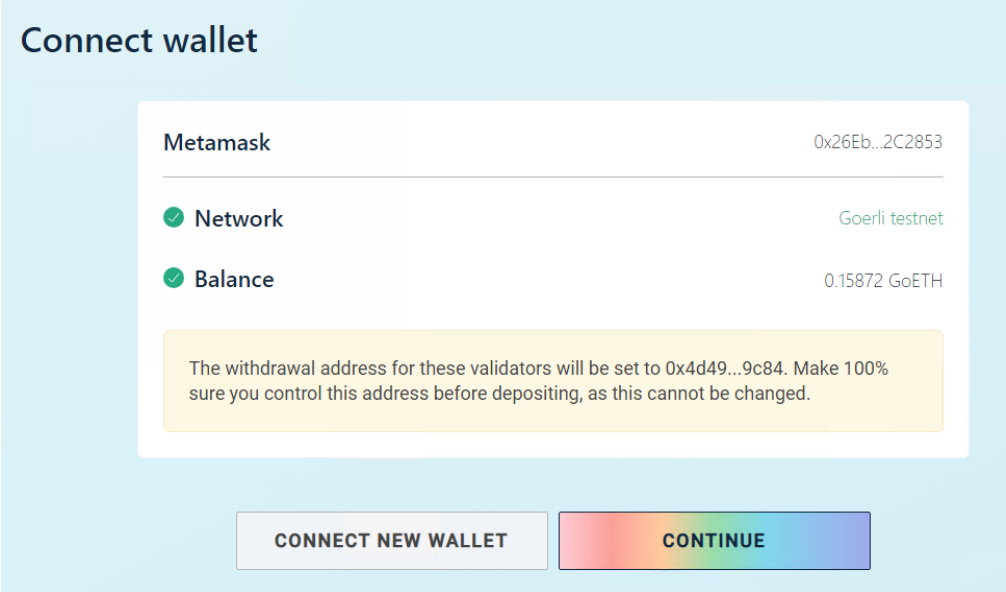
Uploading to the Website

With the deposit data on your computer, go to the website and upload it. This helps the site know you're ready to continue.

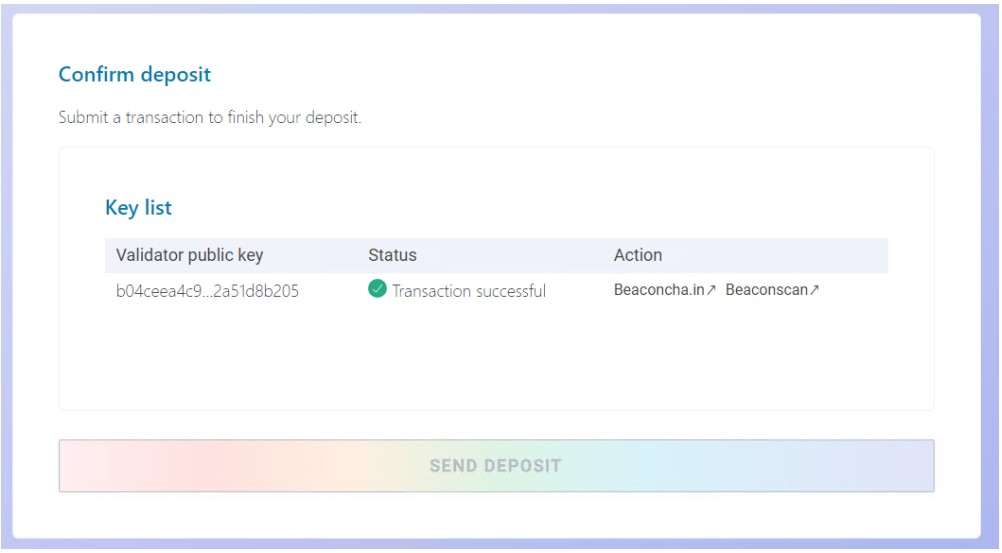


Final Steps

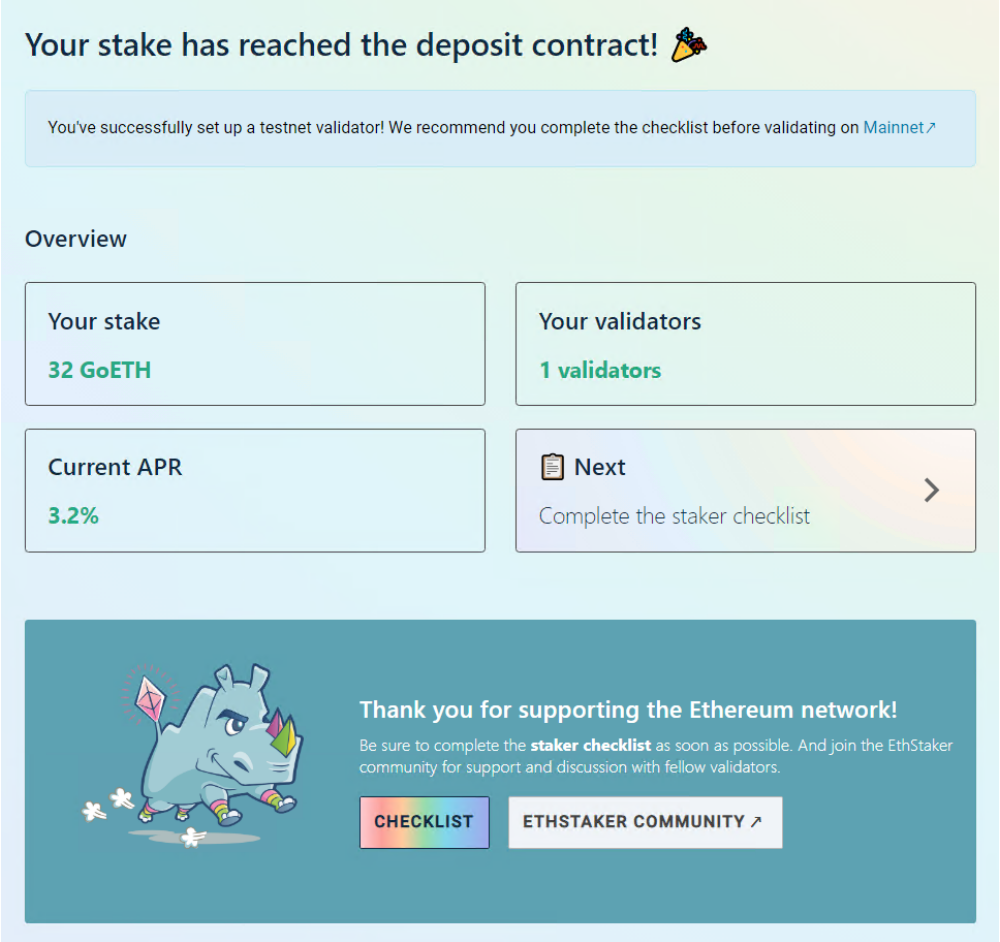
Connect your digital wallet to the website. It'll check which network you're on and how much balance you have.



Go ahead and complete the transaction on the website.



Agent *T* - *"**Congratulations!** You've set up both Execution and Consensus clients, and your staking deposit is ready to go. As soon as your deposit is active, you'll start staking and earning rewards. Great job – you're a tech pro! 🌟"*



10.13- Monitoring your Validator

Checking the status of your validator

Understand the Wait Time

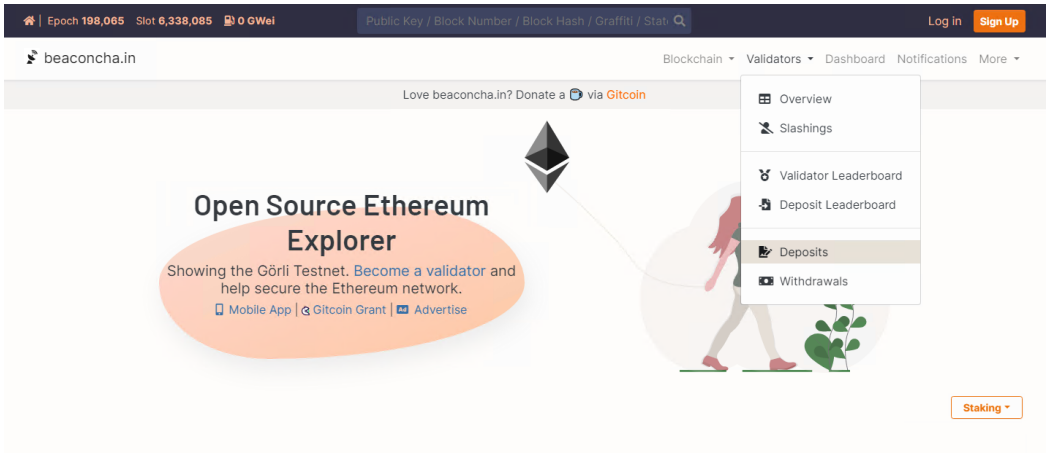
Just added validators can sometimes take a few hours, days, or even weeks to become active.

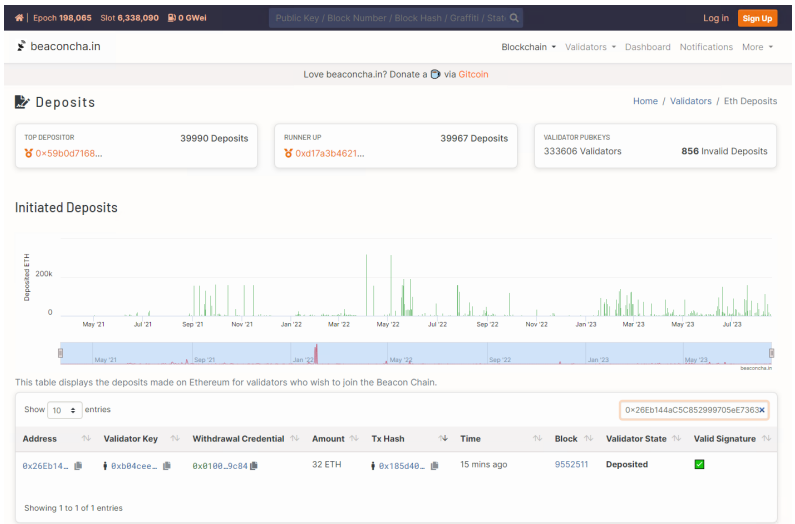
Grab Your Wallet Address

Use your MetaMask to copy the wallet address. Remember, it's the same address you used when you deposited.

Check Your Status

Visit https://goerli.beaconcha.in/
Navigate to the "Deposits" section.
Enter your wallet address to search for your validator key.

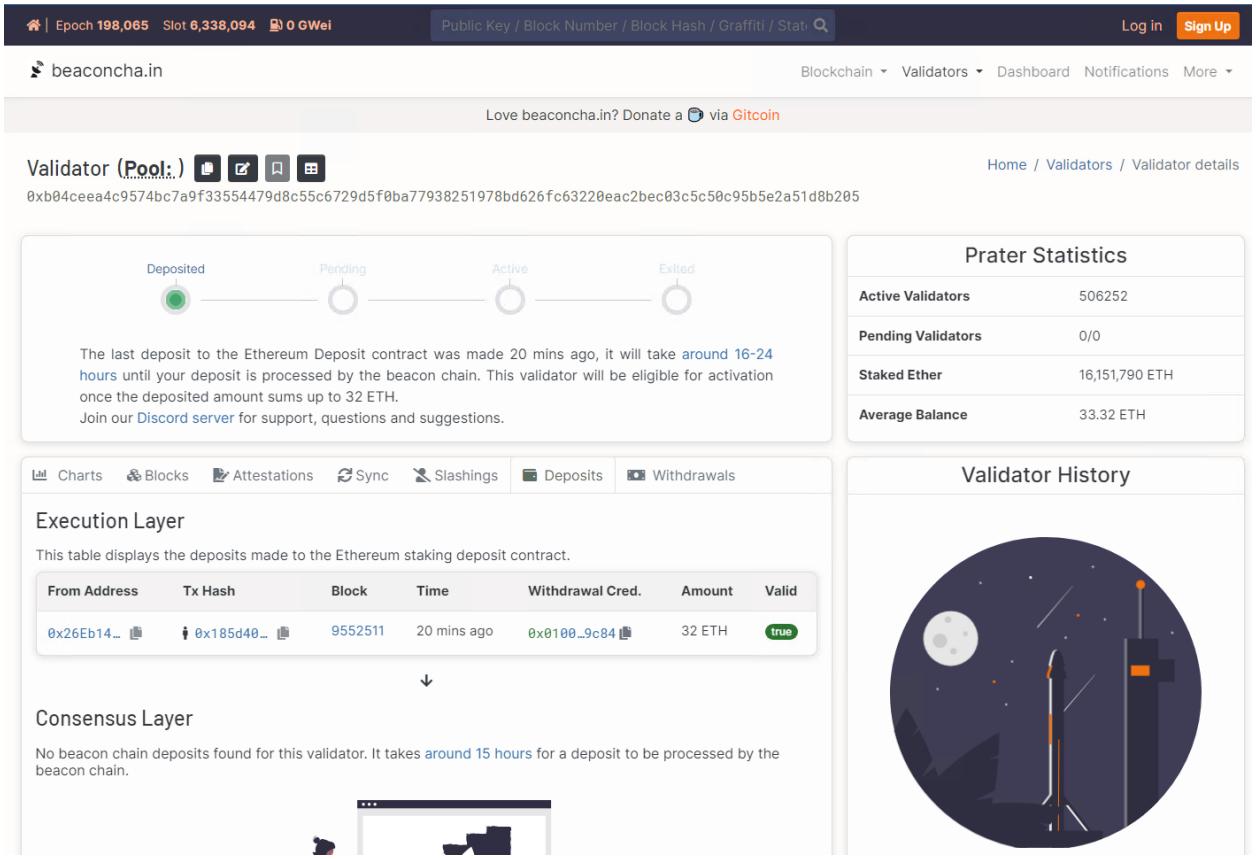




See Your Node's Status

See Your Node's Status

Once you've entered your wallet address, you'll be able to view the state of your node. Typically, it takes around 16-24 hours for the deposit to be processed and your validator to be activated.



Monitoring Validator with Grafana and Prometheus

Grafana and Prometheus are popular tools in the world of computer systems monitoring. When combined, they offer a powerful solution to keep an eye on the performance and health of your Ethereum clients.

Basic Steps to Implement:

Set Up Prometheus

- Install Prometheus on your system.
- Configure it to scrape metrics from your Ethereum client.

Integrate Grafana

- Install Grafana and link it to Prometheus as a data source.
- Design or import dashboards that display the Ethereum client's metrics.

Monitor and Analyze

With everything set up, you can now monitor your Ethereum client's activity in real-time. This helps in ensuring optimal performance and catching potential issues early.

Agent *T* - *"I am aware that these steps are a high-level overview. If there's strong interest from the spy community, I'll delve deeper into this topic and provide a comprehensive, step-by-step explanation in Version 2 of this guide.*

Your feedback is essential, and I want to create resources that are most helpful to you! So while we're working on the second version of our guide, you might find Somer Esat's guide: Ethereum Staking Guide by Somer Esat *extremely helpful. It's well-documented and can serve as an excellent reference point until we release Version 2 of our guide."*

11- Security, Uptime, Performance

Ensuring that node operates efficiently, securely, and continuously, as it's crucial for the integrity and success of the intelligence network.

Fortifying the Station (Security)

- **Encryption Protocols**

Just as a spy would use coded messages, use strong passwords and SSH keys to communicate with and access your station.

- **Secure Perimeter**

Limit access to the station. Only trusted agents with verified credentials should have access.

- **Regular Intel Updates**

Constantly update your station's software and tools, just like spies update their gadgets.

- **Safe Deposit Box**

Like secret documents, store your validator keys in a secure location, such as a safe or a vault.

- **Backup Communication Lines**

Always have backup copies of important messages and data to ensure you never lose valuable intel.

- **Surveillance Systems**

Keep an eye out for enemy activity. If you notice anything unusual, raise the alarm.

Continuous Operation (Uptime)

- **Backup Facilities**

Just as you would have safe houses in different locations, set up backup configurations for uninterrupted operations.

- **Multiple Communication Channels**

Have multiple ways to send and receive intel to prevent being cut off from your network.

- **Early Warning Systems**

Have sensors and alarms to alert you immediately if the station faces threats or goes offline.

- **Discreet Routines**

Regularly inspect your equipment, but do it discreetly to avoid revealing your activities.

Optimal Performance (Performance)

- **Choose the Right Base**

Ensure your station (or Google Cloud instance) has all the resources—like gadgets, tech, and manpower—to handle intel operations efficiently.

- **Regular Surveillance**

Like a spy keeping tabs on the enemy, monitor how your station's resources are used to prevent overloading.

- **Gadget Tuning**

Fine-tune your tools (Ethereum client) for peak performance.

- **Advanced Tech**

Use the best equipment available, akin to using cutting-edge spy tech.

- **Routine Health Checks**

Like regular training sessions, ensure your operations are always in top shape.

Agent T *- "Remember, agent, the integrity of the intelligence network (Ethereum network) depends on the smooth and secure operations of relay stations like yours. Stay vigilant, stay secure, and ensure your station is always operational. Good luck!"*

12- Asset Protection & Withdrawal Accounts

Securing and managing a vault containing priceless assets acquired during covert operations are very important. This vault (akin to the Ethereum withdrawal account) is where the treasures (staking rewards) are stored. You also have a separate locker where the assets are initially held for operations (validator account). It's vital to ensure the treasures are safe, and when the time comes, they can be safely withdrawn and relocated.

Two Separate Lockers (Separation of Validator and Withdrawal Accounts)

- **Compartmentalize**

Always keep your operational assets and the accumulated treasures in separate lockers.

- **Distinct Access**

Use different codes and keys for each locker, making it tough for adversaries to gain unauthorized access.

Fortified Safes (Hardware Wallets or Cold Storage)

- **High-Security Vaults**

Use specially designed safes (hardware wallets) resistant to tampering and breaches.

- **Offline Protection**

Keep your vaults in undisclosed locations away from digital threats.

Vigilance (Regularly Monitor Vault Activity)

- **Routine Checks**

Regularly inspect the vault to ensure the assets are undisturbed and to confirm any additions.

- **Track Movements**

Monitor all deposits and withdrawals, ensuring they align with mission objectives.

Team-Verified Withdrawals (Multi-Signature Solutions)

Ensure that any significant withdrawal requires verification from multiple trusted agents. This eliminates the risk of rogue decisions.

Dry Runs (Testnet and Test Withdrawal Process)

Before moving a large stash of treasures, do a practice run to ensure the process is foolproof and the destination is safe.

State-of-the-Art Security (Keep Software Updated)

Continuously upgrade your vault's security mechanisms, ensuring it’s impervious to the latest breach techniques.

Contingency Plans (Backup Your Keys)

Have duplicate keys and codes stored in secret and secure locations. This ensures you always have access even if the primary key is lost.

Continuous Training (Stay Informed About Security Best Practices)

Stay updated with the latest in espionage and security tactics. Attend spy workshops, get advisories from the agency, and be ahead of the game.

Agent T - *“*Agent, this vault contains assets critical to the agency and the success of its operations. Secure it with your life, and ensure that when the time comes, you can withdraw the assets safely and efficiently. Your agency trusts you to get this right. Good luck!”

With great power comes great responsibility

Young agent,

Congratulations on successfully setting up and running an Ethereum node! Your prowess and dedication have led you to this pivotal moment, marking your entry into an elite league of technologists and spies.

As you embark on this intricate journey into the world of espionage and technology, I want you to remember this: every challenge you face, every node you secure, and every transaction you validate will pave the way for a safer, more resilient Ethereum network.

Upon successful completion of this mission, you'll be actively participating in securing Ethereum and earning rewards for your contributions. Your work will be instrumental in ensuring the network remains an unbreachable fortress.

Good luck, agent! Remember, with great power comes great responsibility. Secure the Ethereum network with diligence and precision. Always stay vigilant, and may your operations be as seamless as the transactions you'll be overseeing.

Until our paths cross again,

Agent T
Head Instructor

This message will self-destruct in:
30... 29... 28... 27... (...)... 3... 2... 1...

[Message Destroyed]

Thank You to the Ethereum Community

🌟 To the EthStaker Team: A big shoutout to the Admins and Educators at EthStaker. Your dedication and expertise play a pivotal role in the community's growth.

🌟 Ethereum's Backbone: Immense gratitude to the Ethereum Execution and Consensus Client teams, core developers, and researchers. Your tireless efforts and innovations keep Ethereum vibrant and evolving.

🌟 Staking Enthusiasts: Thank you to the entire staking community for your valuable feedback, support, and for pushing boundaries.

🌟 Special Mention to Somer Esat: A dedicated note of thanks to Somer Esat for crafting the guide on staking on Ethereum. Your insights helped me dive deep into setting up an Ethereum node on the Google Cloud Platform. Check out his invaluable guide here: Ethereum Staking Guide by Somer Esat.

Disclaimer

⚠️This article is intended for informational purposes only and does not serve as professional advice. The accuracy, integrity, quality, completeness, or timeliness of the information contained herein cannot be guaranteed. The content is presented "as is" and may be subject to updates without prior notice.

⚠️The author disclaims all warranties, express or implied, including but not limited to the accuracy, completeness, and fitness of the information for a specific purpose. The author shall not be held responsible for any direct, indirect, incidental, or consequential damages, including but not limited to personal injuries, business interruptions, data loss, lost profits, or any other financial losses, resulting from the use of or reliance upon the information in this article. This holds true even if the author has been made aware of the potential for such damages.

Contact

Github : https://github.com/Coding4Toon