Preface

Artificial Intelligence (AI) is changing the course of human history. On the back of six decades of Moore's Law and three decades of the internet comprising all of human knowledge, large language models (LLMs) scale at an unprecedented rate towards AGI - by some accounts, arriving as soon as 2027.

Limitless, abundant productivity and creativity brought by AGI will free humanity to pursue our highest potential. AGI is poised to contribute trillions to the global economy annually, transform industries by driving the marginal cost of intelligence toward zero, and morph society by accompanying us as thought partner and companion. This global-scale change could uplift all of humanity.

However, our current computing infrastructure is insufficient to support the development of AGI, and even more alarming is that this immense power is concentrated in the hands of a few.

A global-scale technology revolution requires both a global-scale technological infrastructure and a global-scale alignment mechanism. A world AI infrastructure could leapfrog the confines of data centers and support exponential scaling towards AGI so that humanity could benefit from unlimited intelligence. A global-scale coordination and alignment mechanism ensures that the benefits of AI can be widely, fairly, and safely distributed.

Jan Leike ✔
@janleike · Follow

Replying to @janleike

Building smarter-than-human machines is an inherently dangerous endeavor.

OpenAI is shouldering an enormous responsibility on behalf of all of humanity.

4:57 PM · May 17, 2024

❤ 3.9K Reply Copy link

Read 92 replies

The First Decentralized AI Operating System (deAIOS)

0G is building the global foundation for a better, fairer, and more open AI ecosystem, where power is distributed and innovation thrives.

With 0G deAIOS, the **TOMA** (transparent, owned, monetized, and aligned) future for AGI is near:

- **Transparency**: Centralized AI systems operate as black boxes, with opaque decision-making processes and no clear data provenance. Users who provide training data rarely receive adequate rewards for their contributions, and the system lacks transparency in how value is distributed. *Decentralized AI fosters transparency by providing open, auditable decision-making processes and verifiable data sources, ensuring accountability at every step.* **0G's modular architecture enables transparent data traceability and verifiable AI models, making every decision and data source traceable.**

- **Ownership**: Centralized AI platforms often sell or misuse personal data, with users having little control or ownership over how their information is collected, stored, or used. *Blockchain technology empowers users to retain control over their data, deciding how and when it's used.* **With 0G, users can store and manage data on-chain through decentralized storage networks, retaining full ownership and control.**

- **Monetization**: Centralized AI systems often monopolize profits and leave little incentive for individual contributors. *Blockchain enables the issuance of rewards to incentivize participation and subsequently distribute rewards in a fair and transparent manner.* **Users of 0G's storage, data availability, or serving platforms pay a small fee for these services, and these rewards are then distributed to the nodes.**

- **Alignment**: Centralized AI systems lack oversight and governance processes. A select group may have the authority to censor or promote material, leading to misalignment with human values. *Decentralized AI platforms promote collective governance and oversight by increasing access to AI development, fostering inclusivity and innovation by a broader community rather than a single entity.* **0G provides access to lower-cost compute resources, lowering the high cost barrier to entry for inference, fine-tuning, and training.**

As the first deAIOS, 0G brings together models, compute, and data on a single global platform, enabling developers to build AI applications that are more **PLG** (performant, limitless, and global) than ever before:

- **Performant**: Currently, developers and researchers lack access to high-grade computing and storage resources, limiting their ability to advance at the cutting edge of AI development. *Decentralized GPU networks that run on blockchains provide high-performance compute resources pooled by the community and available to all.* **0G provides access to enterprise-grade compute resources, enabling developers to build AI applications that are more performant than ever before.**

- **Limitless**: Currently, compute clusters and data sets are siloed and limited in scale. *Decentralized blockchain-based compute networks and data sources empower AI developers to scale their dapps beyond the confines of currently available compute and data.* **0G provides infinite scalability and access to compute resources and data from anywhere in the world.**

- **Global**: Currently, only certain regions in the world provide the infrastructure needed to advance AI. *Decentralized AI platforms built on blockchain networks can democratize access to AI development.* **Users of 0G's global AI operating system can build AI applications that are accessible to everyone, everywhere.**

In the following sections, we'll dive deeper into how 0G enables this vision and what it means for the future of AI.

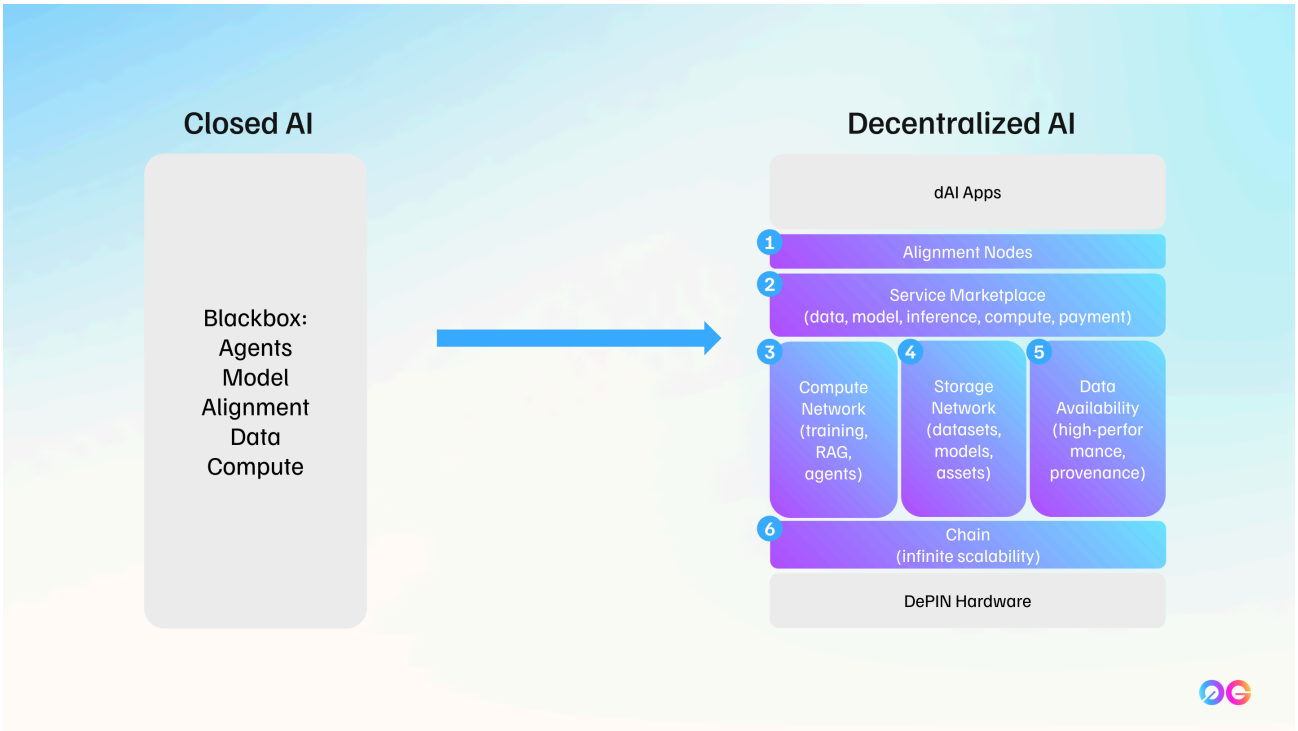
Introduction to 0G

The First Decentralized AI Operating System (deAIOS)

0G (Zero Gravity) is the first decentralized AI operating system that acts as the foundational layer for decentralized AI applications and chains. It efficiently orchestrates utilization of hardware resources such as storage and compute and software assets such as data and models to handle the scale and complexity of AI workloads.

Specifically designed for AI (d)applications and chains, 0G features the fastest, infinitely scalable data availability (DA) layer, the cheapest decentralized storage system, and a novel flexible serving framework for inferences and fine-tuning. Prior to 0G, resource-intensive AI applications would only be feasible "off-chain" due to burdensome on-chain costs, low latency and throughput, and a lack of builder-friendly tooling.

For the first time ever, 0G bridges this cost and efficiency gap between Web2 and Web3, empowering builders to confidently deploy AI applications on-chain with *improved* performance.



The 0G deAIOS has 4 key components:

- **0G Chain**: The fastest, most modular blockchain, making on-chain gaming and AI a reality.
- **0G Storage Network**: A decentralized storage network optimized for massive data loads.
- **0G Compute Network**: A versatile framework for AI compute, including fast data retrieval, model inference/training, etc.
- **0G DA**: An infinitely scalable and programmable data availability layer designed for high-performance chains.

> **What is decentralized storage?**

All of these network components utilize carefully designed and innovative sharding mechanisms to achieve infinite horizontal scalability, thereby removing obstacles to the true democratization of AI.

0G's Namesake

"0G" gets its name from "Zero Gravity," personifying the ultimate goal of achieving a state of weightlessness where transactions and data exchanges can occur effortlessly. Infrastructure should be customizable for developers and invisible to the end-user.

With 0G's technology, the possibilities are endless. Previously infeasible use cases are now within reach, including:

- On-chain AI, including training large language models (LLMs)
- Storage of entire AI agent networks on-chain
- Storage of large datasets for data cleaning and labeling

And this is just the beginning.

Our Mission: Make AI a Public Good

At 0G, our mission is clear: To Make AI a Public Good.

Every component of our ecosystem contributes toward this goal, and we invite you to join us in building the foundation for a decentralized AI future.

This introduction serves as your gateway into the 0G ecosystem. For deeper insights on various products, explore the next sections to learn about 0G deAIOS. If you're ready to dive in, check out our guides:

- Run a Node

- Use our Storage Network
- Learn more about 0G's Architecture

Join the 0G Community

- Discord
- Twitter
- GitHub

We're excited to have you on board as we build the future of Web3xAI infrastructure together!

0G Chain: The Fastest Modular AI Chain

0G is the largest AI Layer 1 ecosystem, powered by the fastest DeAIOS (decentralized AI Operating System). With $325M raised, 0G is the ultimate home for DeAI. Built with a modular architecture, it allows for the independent optimization of key components like consensus, execution, and storage—making it ideal for AI-driven workflows. 0G is fully **EVM-compatible**, so decentralized applications (dApps) already deployed on other L1 or L2 chains (such as Ethereum or rollups) can easily leverage 0G's products without needing to migrate entirely.

0G Chain supports a data availability network, distributed storage network, and AI compute network. These networks integrate with 0G Chain's highly scalable consensus network, which is built to handle massive data volumes suitable for AI.

As the demand for network capacity increases, new consensus networks can be added to enable horizontal scalability, thereby boosting the overall bandwidth and performance of the system. By **decoupling data publication from data storage**, 0G optimizes both throughput and scalability, surpassing the limitations seen in existing data availability (DA) solutions.

0G Chain has 3 unique features:

1. Modular Scalability for AI
2. Custom Consensus
3. Shared Staking

Modular Scalability for AI

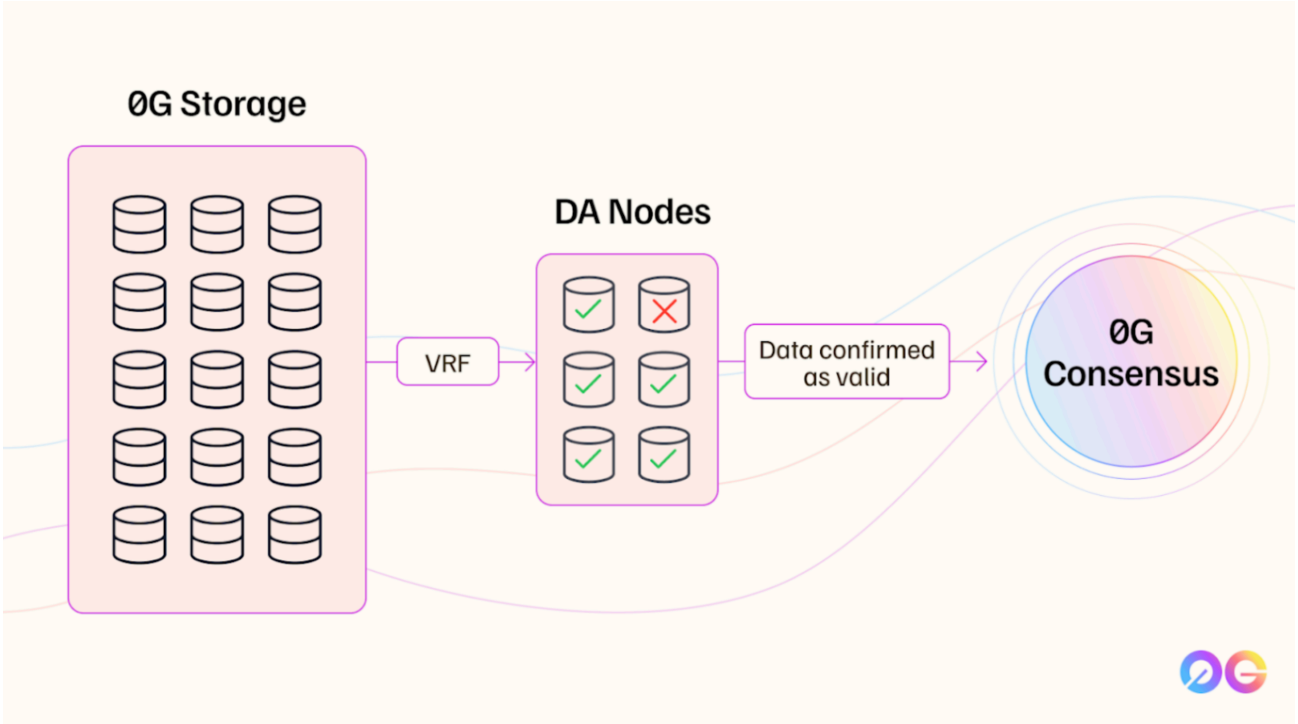
A key differentiator of 0G is its **ability to efficiently manage large volumes of data with exceptional throughput**, thanks to its horizontally scalable architecture. At the core of this architecture is **0G Consensus**, the system's unique consensus mechanism.

0G Chain is also designed with modularity in mind, making it highly adaptable for AI and other data-heavy applications. Its ability to separate the DA layer from the data storage layer allows AI tasks, including large-scale training or inference, to access and store data efficiently. This makes 0G an optimal solution for decentralized AI infrastructures.

0G Consensus: A Distributed Approach

Unlike traditional blockchains that rely on a single, monolithic consensus layer, 0G Consensus is a distributed system composed of multiple independent consensus networks. These networks can expand dynamically based on demand, ensuring that as data volumes grow exponentially, 0G's throughput scales accordingly (e.g. 1, 100, or even 1,000 networks).

This works by having 0G Consensus collaborate with #0G_Storage to validate data. DA Nodes are randomly queried (using a VRF) and collectively reach a consensus on the validity of the data, which is broadcasted to 0G Consensus. As the amount of data to be confirmed grows exponentially, the consensus networks (that in aggregate form 0G Consensus) can expand.



Shared Staking

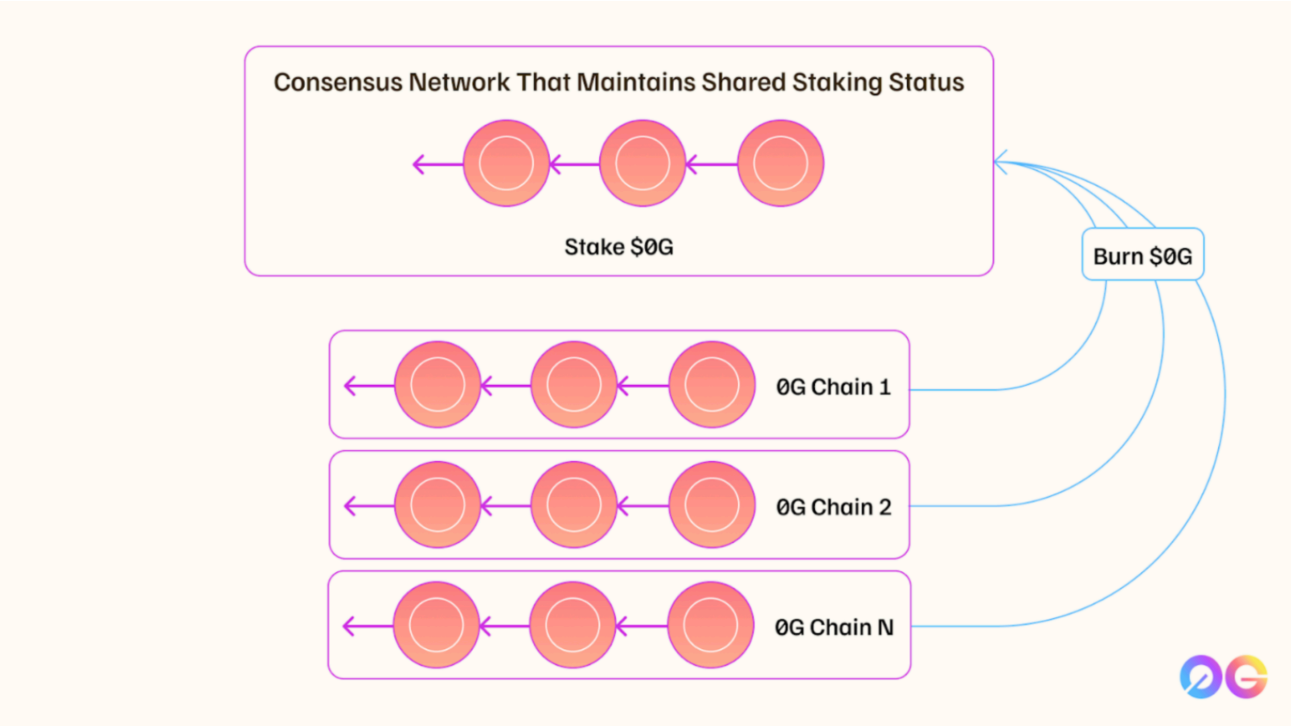
Traditionally, adding more consensus networks would require validators to stake additional assets for each network, making it difficult to maintain sufficient security across the entire system. 0G overcomes this challenge by implementing a **shared staking model**.

In this model, validators stake their funds on the Ethereum mainnet, providing security across all the 0G Consensus networks they participate in. If a slashable event occurs on any of the 0G networks, the validator's stake on the Ethereum

mainnet is subject to slashing. This ensures that **mainnet-level security** is extended to all of 0G's consensus networks simultaneously, creating a unified and secure staking environment.

Reward Mechanism

Validators participating in 0G Consensus earn tokens as rewards for their work. These tokens are burned on the 0G networks, and equivalent tokens are minted on the Ethereum mainnet, where the original funds were staked. This ensures a seamless and efficient reward system that bridges 0G Consensus with Ethereum's robust infrastructure.



Why 0G?

0G is designed with AI applications in mind, offering a highly scalable, modular infrastructure that addresses the unique challenges of data-heavy use cases. Its EVM compatibility ensures that developers already using Ethereum, Layer 2 rollups, or other chains can easily integrate 0G's services—such as DA and storage—without leaving their current platforms. We are also actively exploring the ability to support the Solana VM, Near VM, and BTC compatibility so that AI applications may scale across a broader user base.

0G is not just another Layer 1 chain. It's a modular, AI-optimized system built for the future of decentralized applications. Whether you're working on AI, large-scale data tasks, or any application needing high performance and scalability, 0G provides the tools to grow without compromising on security, speed, or flexibility.

0G Storage: Built for Massive Data

Intro to Storage Systems

Storage systems play a critical role in managing, organizing, and ensuring the accessibility of data. In traditional systems, data is often stored centrally, creating risks around availability, censorship, and data loss due to single points of failure. Decentralized storage addresses these issues by distributing data across a network of nodes, enhancing security, resilience, and scalability. This decentralization is essential, especially in an era where massive datasets are generated and consumed by AI, Web3 applications, and large-scale businesses.

0G's Storage System

0G Storage is a distributed data storage system designed with on-chain elements to incentivize storage nodes to store data on behalf of users. Anyone can run a storage node and receive rewards for maintaining one. For information on how to do so, check out our guide here.

0G's system consists of two parts:

1. **The Data Publishing Lane:** Ensures data availability by allowing quick queries and verification through the 0G Consensus network. This ensures that stored data can be easily accessed and validated by users or applications when needed.
2. **The Data Storage Lane:** Manages large data transfers and storage using an erasure-coding mechanism. This splits data into smaller, redundant fragments distributed across different nodes, guaranteeing data recovery even if some nodes fail or experience downtime.

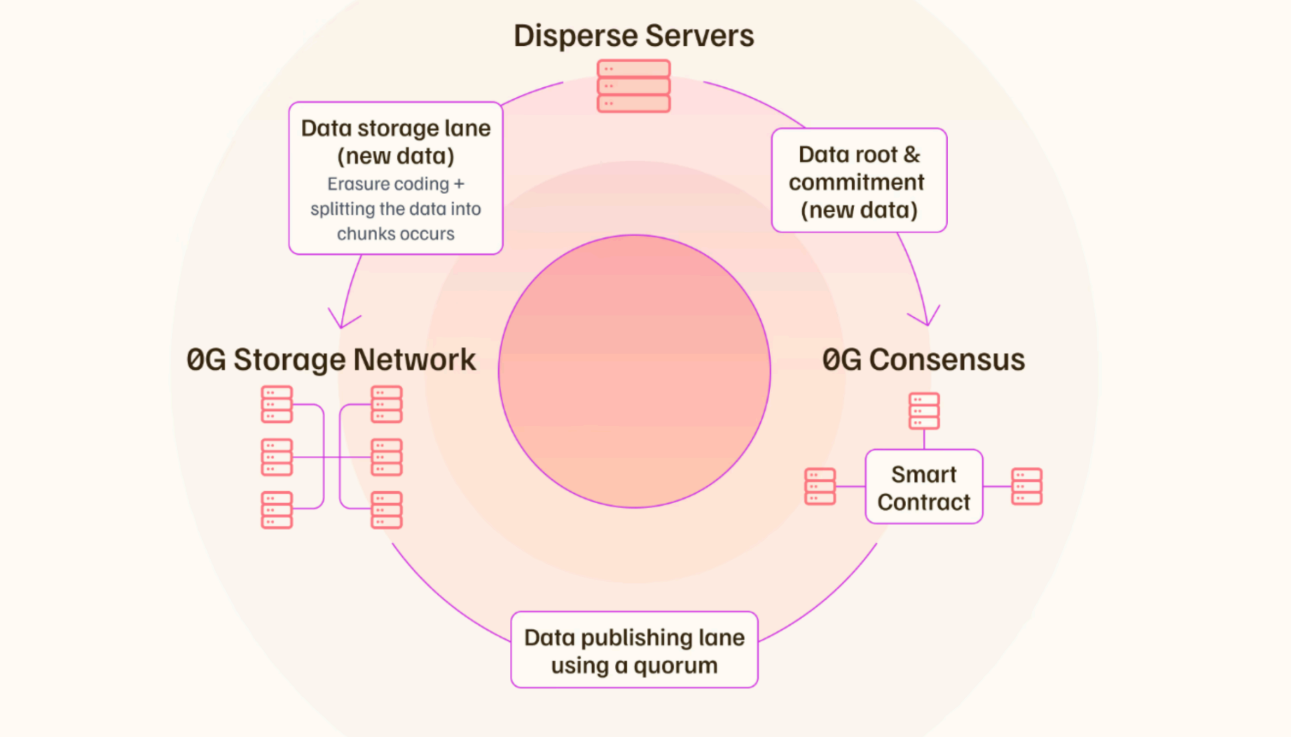
For any party wishing to store data with 0G, the data must first be provided alongside payment using 0G's token, which is fully embedded into 0G's main chain. To store this data, it is first *erasure-coded,* meaning that the data being stored is fragmented into redundant smaller pieces distributed across multiple storage locations. Redundancy is essential as it ensures the data can always be recovered, even if some storage locations fail or become unavailable.

0G system also has "disperse servers" that handle:

- Erasure coding the data and distributing it across multiple storage nodes.

- Generating a **data root**, which is essentially the Merkle root of the encoded data chunks. This root serves as a **commitment** to the integrity and completeness of the data. A Merkle root is a cryptographic hash that is formed by combining hashes from various individual transactions.

0G's storage infrastructure is designed to support heavy data loads, making it a strong candidate for applications that demand high throughput and reliability. With separate consensus networks managing distinct storage partitions (covered in #0G_CHAIN), 0G's storage system scales effortlessly, addressing the needs of users with data needs of any size.



The 0G Storage Architecture

0G Storage is built on a layered architecture designed to accommodate both unstructured and structured data, ensuring flexibility and scalability for a wide range of applications.

- **The Log Layer**: Handles unstructured data and functions similarly to a traditional computer file system. This layer is ideal for storing large-scale data storage where the data does not change (e.g. archival purposes, data logs) as it is an append-only system, meaning new data can be added but not modified.
- **The Key-Value (KV) Layer**: Built on top of the Log Layer, the Key-Value Layer is optimized for structured data. Unlike the log layer, this layer is mutable and can be updated via new entries that are added to log entries, allowing for efficient storage and retrieval of key-value pairs. This layer is best for dynamic

applications that require frequent updates, such as databases or collaborative documents (e.g. an on-chain version of Notion or Google Docs).

Both layers are secured and validated through 0G's Consensus, which guarantees the accuracy and availability of all stored data.

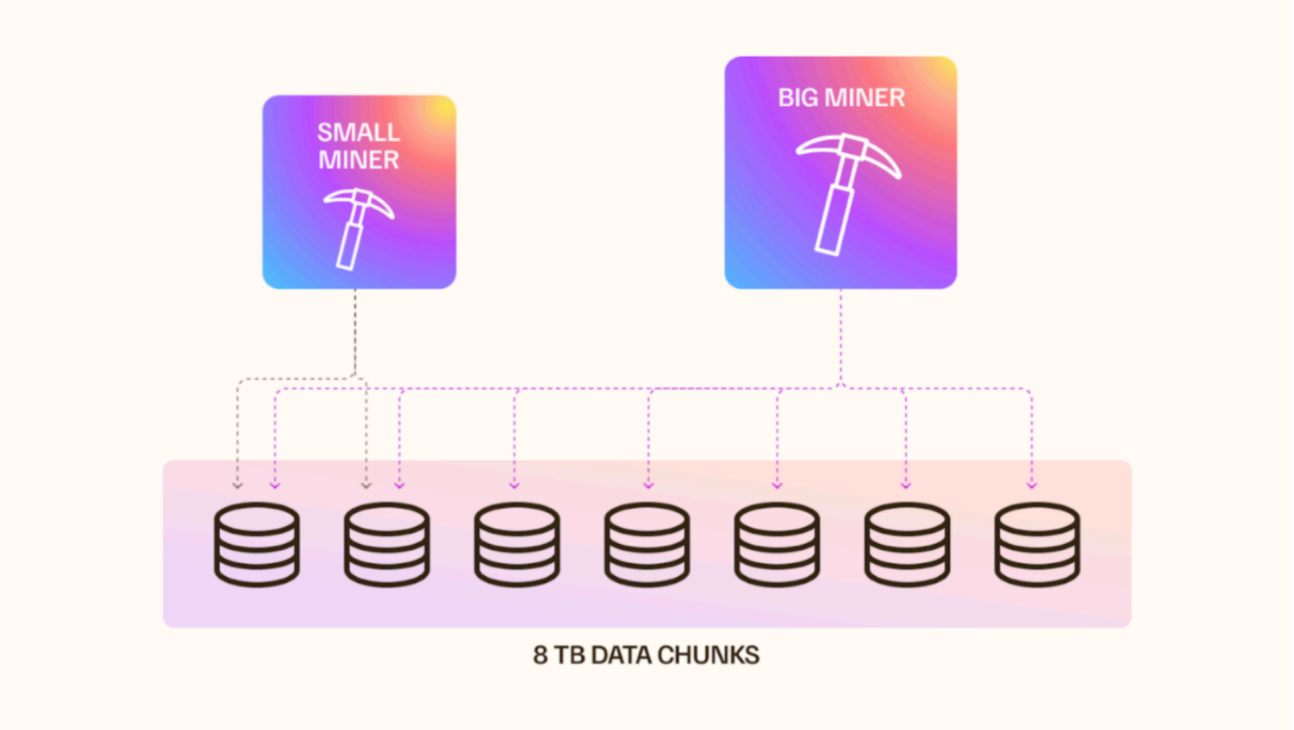
Proof of Random Access

0G Storage is maintained by a network of miners incentivized to store and manage data through a unique consensus mechanism known as **Proof of Random Access (PoRA)**.

Miners are periodically challenged to confirm that they are actively storing specific data chunks. This process works similarly to Proof of Work (PoW), where miners must provide a valid cryptographic hash to verify they are storing the correct data. Before a miner can respond to a query, they must be selected by generating a satisfactory cryptographic output: typically, a hash with a certain number of leading zeros.

To promote fairness, the mining range is capped at 8 TB of data per mining operation. This ensures that smaller-scale miners with fewer machines can compete on an even playing field with larger operators.

Miners with more resources can still participate by dividing their machines across multiple 8 TB ranges, allowing them to mine simultaneously in various data segments. Meanwhile, smaller miners can focus on a single 8 TB range, ensuring that the process remains competitive and inclusive for all participants, regardless of scale.



Comparison to Filecoin and Arweave

Decentralized storage systems like Filecoin and Arweave face significant limitations surrounding structured data, scalability, cost, and data availability functionality.

These can be summarized as follows:

- **Unstructured Data Only:** Both of these systems are optimized only for unstructured data, limiting their versatility. In contrast, 0G supports both structured and unstructured data, making it suitable for a wider range of applications, including complex AI workflows and their vast databases.
- **Scalability:** Filecoin or Arweave struggle to handle data at a massive scale. In contrast, 0G partitions both its data storage and network throughput to enable scalability that meets the massive data requirements associated with on-chain AI.
- **Cost:** Benchmarking has shown that permanent storage with Arweave can cost up to $17,000 per TB, while 0G offers a much more cost-effective solution at just $10-11 per TB.
- **Data Availability:** 0G's integrated data availability layer provides ultra-fast, infinitely scalable access to data, enabling a fully vertically integrated solution that supports high-performance applications like decentralized AI.

All in all, 0G Storage is the ultimate solution for any on-chain data storage needs, regardless of whether a project is in the AI space or not.

0G Compute Network: Decentralized Inference & Beyond

In today's world, AI models are transforming industries, driving innovation, and powering new applications. Despite their growing value, there's a significant gap in how AI services are delivered. Centralized platforms often limit access, increase costs, and restrict the flexibility of AI developers. This is where the 0G Compute Network steps in to make a difference.

What is 0G Compute Network?

The 0G Compute Network is a decentralized framework that provides AI computing capabilities to our community. It forms a crucial part of deAIOS and, together with the storage network, offers comprehensive end-to-end support for dApp development and services.

The first iteration focuses specifically on decentralized settlement for inference, connecting buyers (who want to use AI models) with sellers (who run these models on their GPUs) in a trustless, transparent manner. Sellers, known as service providers, can set the price for each model they support and receive real-time rewards for their contributions. It's a fully decentralized marketplace that eliminates the need for intermediaries, redefining how AI services are accessed and delivered by making them cheaper, more efficient, and accessible to anyone, anywhere.

How does it work?

The 0G Compute Network contract facilitates secure interactions between users (AI buyers) and service providers (GPU owners running AI models), ensuring smooth data retrieval, fee collection, and service execution. Here's how it works:

1. **Service Provider Registration:** Service providers first register the type of AI service they offer (e.g., model inference) and set pricing for each type within the smart contract.
2. **User Pre-deposits Fees:** When a user wants to access a service, they pre-deposit a fee into the smart contract associated with the selected service provider. This ensures that funds are available to compensate the service provider.

3. **Request and Response System:** Users send requests for AI inference, and the service provider decides whether to respond based on the sufficiency of the user's remaining balance. Both the user and the provider sign each request and response, ensuring trustless verification of transactions.

Here are some of the key features of the system:

- **Open Access with Fair Rewards:** Anyone with the right hardware can become a service provider and earn fair compensation for running AI models. This open-access, decentralized structure enables a global network of contributors, where providers are directly rewarded for their computational resources and services, fostering a new ecosystem of decentralized AI.
- **Optimized Efficiency:** 0G Compute Network uses a variety of different mechanisms to minimize costs and maximize performance. Service providers can batch-process multiple user requests to minimize the number of on-chain settlements, optimizing transaction costs and network efficiency. ZK-proofs are used to compress transaction data, lowering on-chain settlement costs. Additionally, to reduce the on-chain costs of storing request traces with data keys, 0G Storage allows for scalable off-chain data management, enabling more efficient storage and retrieval while keeping costs low.
- **User-Centric Design:** The platform offers a smooth user experience, with a built-in refund mechanism that ensures users can reclaim unused funds within a clearly defined time window. This process is executed by smart contracts, ensuring a reliable, secure, and frictionless process for both service providers and users.

By decentralizing both services and settlement, 0G Compute Network provides a scalable and trustless alternative to centralized AI platforms.

Over time, we aim to decentralize the entire AI workflow—from inference to data and training—by keeping everything on-chain and autonomous.

0G DA: Infinitely Scalable and Programmable DA

The Rise of Data Availability Layers

Data availability (DA) refers to proving that data is readily accessible, verifiable, and retrievable. For example, Layer 2 rollups such as Arbitrum or Base reduce the burden on Ethereum by handling transactions off-chain and then publishing the data back to Ethereum, thereby freeing up L1 throughput and reducing gas costs. The transaction data, however, still needs to be made available so that anyone can validate or challenge the transactions through fraud proofs during the challenge period.

As such, DA is crucial to blockchains as it allows for full validation of the blockchain's history and current state by any participant, thus maintaining the decentralized and trustless nature of the network. Without this, validators would not be able to independently verify the legitimacy of transactions and blocks, leading to potential issues like fraud or censorship.

This led to the emergence of Data Availability Layers (DALs), which provide a significantly more efficient manner of storing and verifying data than publishing directly to Ethereum. DALs are critical for several reasons:

- **Scalability**: DALs allow networks to process more transactions and larger datasets without overwhelming the system, reducing the burden on network nodes and significantly enhancing network scalability.
- **Increased Efficiency**: DALs optimize how and where data is stored and made available, increasing data throughput and reducing latency while also minimizing associated costs.
- **Interoperability & Innovation**: DALs that can interact with multiple ecosystems enable fast and highly secure interoperability for data and assets.

However, it's worth noting that not all DALs are built equally.

The Challenge Today

Existing DALs tend to require that data be simultaneously sent to all of their network nodes, preventing horizontal scalability and limiting network speed to its slowest node. They also do not have built-in storage systems, requiring connectivity to external systems that impact throughput, latency, and cost.

Additionally, 0G inherits Ethereum's security, while other systems rely upon their own security mechanisms that fall short. This is significant because Ethereum's network is secured by over 34 million ETH staked, representing approximately $80 billion in cryptoeconomic security at the time of writing. In contrast, competitors rely on security mechanisms that, at best, represent only a fraction of Ethereum's total security. This gives 0G a distinct advantage, as it leverages the vast economic incentives and decentralization of Ethereum's staking system, providing a level of protection that competitors cannot match.

Even more issues exist, including EigenDA's lack of randomization over its data committees. As data committees are core to a DA system's integrity, a lack of randomization means that collusion is theoretically possible for malicious nodes to predict when they might be on a committee together.

0G's core differentiation is massive throughput and scalability.

This is possible through 0G's unique design includes a built-in storage system and horizontally scalable consensus design, alongside other clever design mechanisms that we'll cover below.

The result is that 0G serves as the foundational layer for decentralized AI applications, bringing on-chain AI and more to life.

Why 0G

There are 4 differentiators of 0G worth highlighting:

1. **Infinitely Scalable DA:** 0G's infinitely scalable DAL can quickly query or confirm data as valid, whether data is held by 0G Storage, or external Web2 or Web3 databases. Infinite scalability comes from the ability to continuously add new consensus networks, supporting workloads that far surpass the capacity of existing systems.

2. **Modular and Layered Architecture**: 0G's design decouples storage, data availability, and consensus, allowing each component to be optimized for its specific function. Data availability is ensured through redundancy, with data distributed across decentralized Storage Nodes. Cryptographic proofs (like Merkle trees and zk-proofs) verify data integrity at regular intervals, automatically replacing nodes that fail to produce valid proofs. And combined with 0G's ability to keep adding new consensus networks that scale with demand, 0G can scale efficiently and is ideal for complex AI workflows and large-scale data processing.

3. **Decentralized AI Operating System & High Throughput**: 0G is the first decentralized AI operating system (deAIOS) designed to give users control over

their data, while providing the infrastructure necessary to handle the massive throughput demands of AI applications. Beyond its modular architecture and infinite consensus layers, 0G achieves high throughput through parallel data processing, enabled by erasure coding, horizontally scalable consensus networks, and more. With a demonstrated throughput of 50 Gbps on the Galileo Testnet, 0G seamlessly supports AI workloads and other high-performance needs, including training large language models and managing AI agent networks.

These differentiators make 0G uniquely positioned to tackle the challenges of scaling AI on a decentralized platform, which is critical for the future of Web3 and decentralized intelligence.

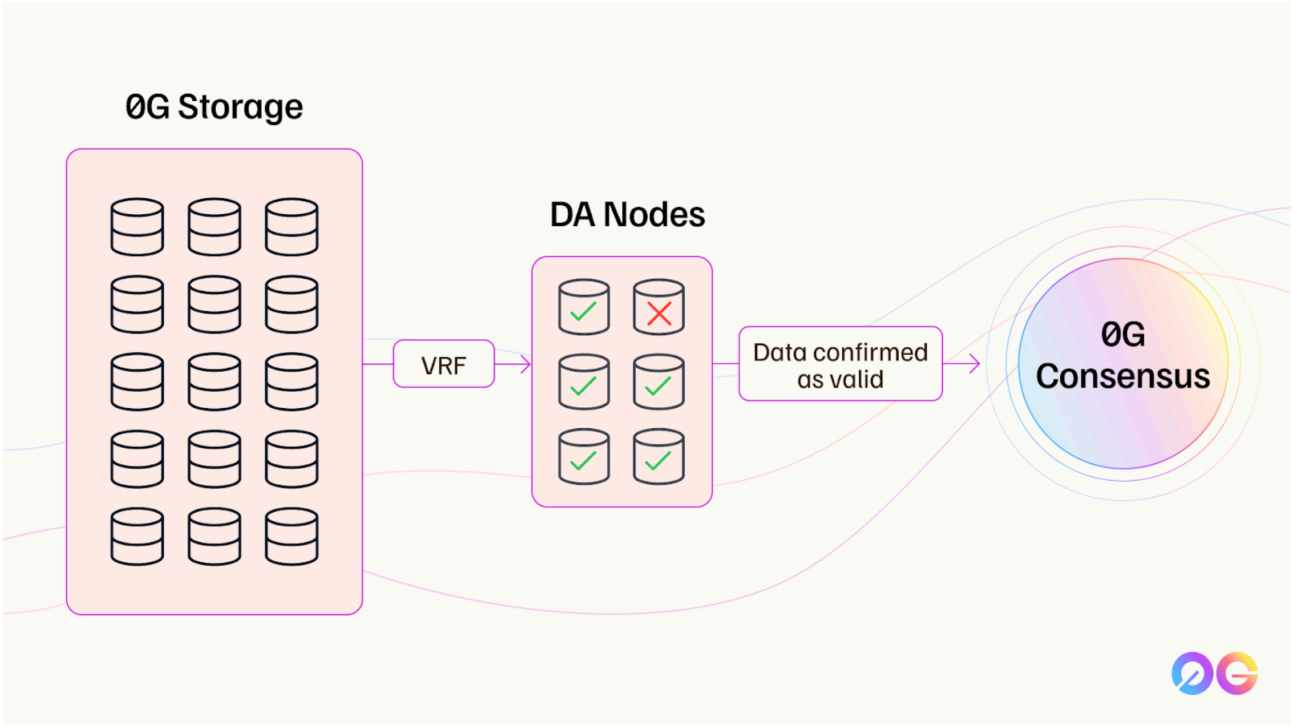
How Does This Work?

As covered in 0G Storage, data within the 0G ecosystem is first erasure-coded and split into "data chunks," which are then distributed across various Storage Nodes in the 0G Storage network.

To ensure data availability, 0G uses **Data Availability Nodes** that are randomly chosen using a Verifiable Random Function (VRF). A VRF generates random values in a way that is unpredictable yet verifiable by others, which is important as it prevents potentially malicious nodes from collusion.

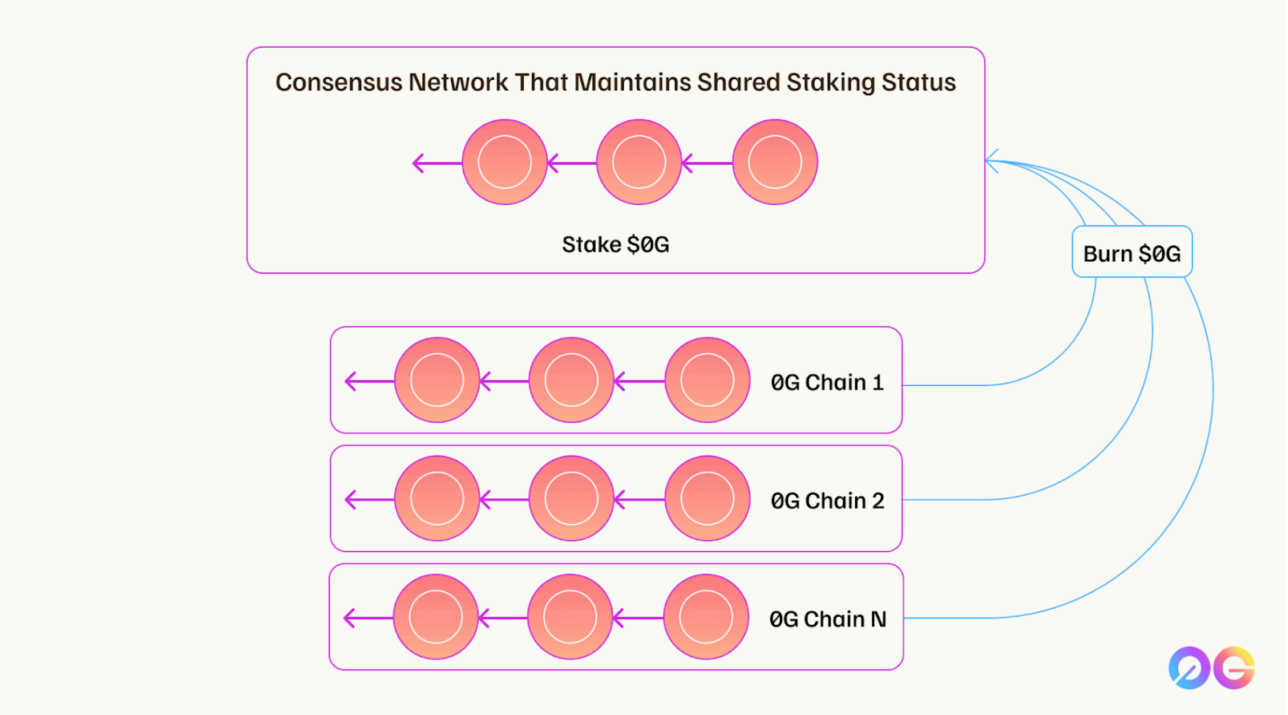
These DA nodes work together in small groups, called quorums, to check and verify the stored data. The system assumes that most nodes in each group will act honestly, known as an "honest majority" assumption.

The consensus mechanism used by 0G is fast and efficient due to its sampling-based approach. Rather than verifying all data, DA nodes sample portions of it, drastically reducing the data they need to handle. Once enough nodes agree that the sampled data is available and correct, they submit availability proofs to the 0G Consensus network. This lightweight, sample-driven approach enables faster verification while maintaining strong security.



Validators in the 0G Consensus network, who are separate from the DA nodes, verify and finalize these proofs. Although DA nodes ensure data availability, they do not directly participate in the final consensus process, which is the responsibility of 0G validators. Validators use a shared staking mechanism where they stake 0G tokens on a primary network (likely Ethereum). Any slashable event across connected networks leads to slashing on the main network, securing the system's scalability while maintaining robust security.

This is a key mechanism that allows for the system to scale infinitely while maintaining data availability. In return, validators engaged in shared staking receive 0G tokens on any network managed, which can then be burnt in return for 0G tokens on the mainnet.



Getting started: link to guide

Use Cases

0G DA offers an infinitely scalable and high-performance DA solution for a wide range of applications across Web3, AI, and more.

L1s / L2s

Layer 1 and Layer 2 chains can utilize 0G DA to handle data availability and storage requirements for decentralized AI models, large datasets, and on-chain applications. Existing partners include networks like **Polygon, Optimism, Arbitrum, Fuel, Manta Network**, **and countless more,** which leverage 0G's scalable infrastructure to store data more efficiently and support fast retrieval.

Decentralized Shared Sequencers

Decentralized Shared Sequencers help order L2 transactions before final settlement on Ethereum. By integrating 0G DA, shared sequencers can streamline data across multiple networks in a decentralized manner, unlike existing sequencers which are often centralized. This also means fast and secure data transfers between L2s.

Bridges

Cross-chain bridges benefit from 0G DA's scalable storage and data availability features. Networks can store and retrieve state data using 0G DA, making state

migration between networks faster and more secure. For example, a network can confirm a user's assets and transfer them securely to another chain using 0G's highly efficient data verification.

Rollups-as-a-Service (RaaS)

0G DA can serve as a reliable DA solution for RaaS providers like **Caldera and AltLayer**, enabling seamless configuration and deployment of rollups. With 0G DA's highly scalable infrastructure, RaaS providers can ensure the secure availability of data across multiple rollups without compromising performance.

DeFi

0G's DA infrastructure is ideally suited for DeFi applications that require fast settlement and high-frequency trading. For example, by storing order book data on 0G, DeFi projects can achieve faster transaction throughput and enhanced scalability across L2s and L3s.

On-Chain Gaming

On-chain gaming platforms rely on cryptographic proofs and metadata related to player assets, actions, and scores. 0G DA's ability to handle large volumes of data securely and efficiently makes it an optimal solution for gaming applications that require reliable data storage and fast retrieval.

Data Markets

Web3 data markets can benefit from 0G DA by storing datasets on-chain. The decentralized storage and retrieval capabilities of 0G enable real-time updates and querying of data, providing a reliable solution for data market platforms.

AI & Machine Learning

0G DA is particularly focused on supporting decentralized AI, allowing full AI models and vast datasets to be stored and accessed on-chain. This infrastructure is essential for advanced AI applications that demand high data throughput and availability, such as training large language models (LLMs) and managing entire AI agent Networks.

0G DA Technical Deep Dive

The Data Availability (DA) module allows users to submit a piece of data, referred to as a ***DA blob***. This data is redundantly encoded by the client's proxy and divided into several slices, which are then sent to DA nodes. ***DA nodes*** gain eligibility to verify the correctness of DA slices by staking. Each DA node verifies the integrity and correctness of its slice and signs it. Once more than 2/3 of the aggregated signatures are on-chain, the data behind the related hash is considered to be published decentrally.

To incentivize DA nodes to store the signed data for a period, the signing process itself does not provide any rewards. Instead, rewards are distributed through a process called ***DA Sampling***. During each DA Sample round, any DA slice within a certain timeframe can generate a lottery chance for a reward. DA nodes must store the corresponding slice to redeem the lottery chance and claim the reward.

The process of generating DA nodes is the same as the underlying chain's PoS process, both achieved through staking. During each DA epoch (approximately 8 hours), DA nodes are assigned to several quorums. Within each quorum, nodes are assigned numbers 0 through 3071. Each number is assigned to exactly one node, but a node may be assigned to multiple quorums, depending on its staking weight.

DA Processing Flow

DA takes an input of data up to 32,505,852 bytes in length and processes it as follows:

1. **Padding and Size Encoding:**
    - Pad the data with zeros until it reaches 32,505,852 bytes.
    - Add a little-endian format 4-byte integer at the end to indicate the original input size.
2. **Matrix Formation:**
    - Slice the padded data into a 1024-row by 1024-column matrix, filling each row consecutively, with each element being 31 bytes.
    - Pad each 31-byte element with an additional 1-byte zero, making it 32 bytes per element.
3. **Redundant Encoding:**
    - Expand the data to a 3072-row by 1024-column matrix using redundancy coding.
    - Calculate the ***erasure commitment*** and ***data root*** of the expanded matrix.
4. **Submission to DA Contract:**

    - Submit the erasure commitment and data root to ***the DA contract*** and pay the fee.
    - The DA contract will determine the epoch to which the data belongs and assign a quorum.
5. **Data Distribution:**
    - Send the erasure commitment, data root, each row of the matrix, and necessary proofs of correctness to the corresponding DA nodes.
6. **Signature Aggregation:**
    - More than 2/3 of the DA nodes sign the erasure commitment and data root.
    - Aggregate the signatures using the BLS signature algorithm and submit the aggregated signature to the DA contract.

Details of erasure encoding

After matrix formation, each element is processed into a 32-byte data unit, which can be viewed interchangeably as either 32 bytes of data or a 256-bit little-endian integer. Denote the element in the i-th row and j-th column as $c_{i,j}$.

Let the finite field F be the scalar field of the BN254 curve. Each element $c_{i,j}$ is also considered an integer within the finite field F. Let p be the order of F, a known large number that can be expressed as $2^{28} \times A + 1$, where A is an odd number. The number 3 is a generator of the multiplicative group of the F. Define $u = 3^{2^6 \times A}$ and $w = 3^{2^8 \times A}$, so $w^{2^{20}} = 1$ and $u^4 = w$.

Now we define a polynomial f over $\mathsf{F} \to \mathsf{F}$ with degree $d = 2^{20} - 1$ satisfying

$$\forall\, 0 \le i < 1024,\ 0 \le j < 1024,\ f(w^{1024j+i}) = c_{i,j}$$

Then we extend the 1024×1024 matrix into 1024×3072 matrix, where

$$\forall\, 1024 \le i < 2048,\ 0 \le j < 1024,\ c_{i,j} = f(u^2 \cdot w^{1024j+(i-1024)})$$

$$\forall\, 2048 \le i < 3072,\ 0 \le j < 1024,\ c_{i,j} = f(u \cdot w^{1024j+(i-2048)})$$

The **erasure commitment** is the KZG commitment of f, defined as $f(\tau) \cdot G$, where G is the starting point of BN254 G1 curve, and τ is a latent parameter from the perpetual powers of tau trusted setup ceremony.

The **data root** is defined as the input root by treating the 1024*3072 32-byte elements as a continuous storage submission input. Specifically, according to the storage submission requirements, these data does not need to pad any zeros, and will be divided into a 16384-element sector array and an 8192-element sector array.

DA nodes need to verify two parts:

1. The consistency between the received slice and the data root, mainly achieved through Merkle proofs.
2. The consistency between the received slice and the erasure commitment, verified using KZG proofs. Here, we use the AMT protocol optimization introduced in LVMT to reduce the proving overhead.

DA Sampling

The blockchain will periodically release DA Sampling tasks at preset height every `SAMPLE_PERIOD` blocks, with the parent block hash of these heights used as the `sampleSeed` for DA Sampling.

List of Parameters

Constant parameters

Parameter	Requirement	Default value
MAX_PODAS_TARGET		2^256 / 128 - 1

Admin adjustable parameters

Parameter	Requirement	Default value	
TARGET_SUBMITS		20	https://github.com/0glab contract/blob/3951565fb
EPOCH_WINDOW_SIZE		300 (about 3 months)	https://github.com/0glab contract/blob/3951565fb
SAMPLE_PERIOD		30 (about 1.5 minutes)	https://github.com/0glab contract/blob/3951565fb

Responses

During each period, each DA slice (one row) can be divided into 32 sub-lines. For each sub-line, the `podasQuality` will be computed using the `dataRoot` and assigned `epoch` and `quorumId` of its corresponding DA blob.

> 🔢 By default, all integers are in 256-bit big-endian format when computing hash values. `lineIndex` is the only exception, which is in 64-bit big-endian format.

The hash value can be viewed interchangeably as either 32 bytes of data or a 256-bit big-endian integer.

```
lineQuality = keccak256(sampleSeed, epoch, quorumId, dataRoot,
lineIndex);
dataQuality = keccak256(lineQuality, sublineIndex, data);
podasQuality = lineQuality + dataQuality
```

If the `podasQuality` is less than the current `podasTarget` in the DA contract and the `epoch` falls within `[currentEpoch - EPOCH_WINDOW_SIZE, currentEpoch)`, then this sub-line is regarded as a ***valid DAS response*** and is eligible for the reward. The DA node assigned to this row can claim the reward.

During a sample period, at most `TARGET_SUBMITS × 2` DAS responses can be submitted and rewarded; any submissions exceeding this limit will be rejected.

Difficulty Adjustment

`TARGET_SUBMITS` valid responses are expected in a sample period. If more or fewer responses are submitted during a sample period, the `podasTarget` will be adjusted as follows:

```
podasTarget -= podasTarget * (actualSubmits - TARGET_SUBMITS) /
TARGET_SUBMITS / 8
```

Economic Model

List of Parameters

Admin adjustable parameters

Parameter	Requirement	Default value	
BASE_REWARD		0	https://github.com/0gla contract/blob/3951565
BLOB_PRICE		0	https://github.com/0gla contract/blob/3951565

Parameter	Requirement	Default value	
SERVICE_FEE_RATE_BP		0	https://github.com/0gla contract/blob/3951565
REWARD_RATIO	[1]	1,200,000	https://github.com/0gla contract/blob/3951565

[1] `TARGET_SUBMITS` × Time elapsed for `EPOCH_WINDOW_SIZE` epochs / Time elapsed in `SAMPLE_PERIOD` / `REWARD_RATIO` should be approximately 0.5 to 2.

Pricing

When users submit the metadata for a DA blob, they need to pay a fee in amount of `BLOB_PRICE`.

Reward

When a DA epoch ends, all the rewards from that DA epoch will be stored in the DA reward pool. Each time a valid response is submitted, `1 / REWARD_RATIO` of the reward pool will be distributed to the corresponding DA node.

System Rewards

In the early stages of the ecosystem, the foundation can reserve a portion of tokens for system rewards. When the DA node submits a valid response, an additional reward of `BASE_REWARD` will be issued.

The funds for the base reward will be manually deposited into the reward contract and tracked separately. If the balance for the base reward is insufficient to cover a single base reward, miners will not be able to receive the full base reward.

Service Fee

A system service fee is charged as a proportion of the DA fees paid by the user, according to the parameter `SERVICE_FEE_RATE_BP`.

Run a node

See here for instructions to become DA signer and run your own node.

Overview

Want to become an active participant in the 0G network and earn rewards while you're at it?

Each node type plays a crucial role in maintaining the 0G network's functionality, from transaction validation and data storage to ensuring data availability and retrieval. Here, we'll introduce you to the various types of nodes you can run, each contributing to the network's health and security.

What Nodes Can I Run?

Validator Nodes

The guardians of the network, validator nodes are responsible for verifying transactions, ensuring consensus, and maintaining the blockchain. They're essential for keeping the 0G blockchain secure and running smoothly.

Storage Nodes

Unlike Validator Nodes that focus on securing the blockchain itself, Storage Nodes focus on managing and serving data. They are the backbone of the network's data storage capabilities, ensuring persistence and availability for long-term data storage (e.g., training datasets, large AI models). By running a storage node, you'll contribute to the decentralized storage of 0G data, making it accessible and resilient.

Data Availability Services

DA Nodes are similar to Storage Nodes but focus on immediacy and short-term accessibility to support real-time operations. This data is typically used by Layer 2 and rollup solutions for data availability and is not typically stored long-term. Think of these nodes as the network's librarians, ensuring that data can be quickly retrieved when needed.

Why Run a Node?

Running a node isn't just about supporting the network; it's also a way to earn rewards for your contribution. By actively participating in the 0G ecosystem, you'll be eligible to receive rewards that incentivize your efforts.

Ready to Dive In?

We've made it easy to get started. The table below outlines the hardware requirements for each type of node, so you can choose the one that best suits your

setup. Once you're ready, head over to the 0G documentation for detailed instructions on how to set up and run your chosen node.

Node Type	Description	Memory	CPU	Disk	Bandwidth
Validator Node	Validates transactions and maintains network consensus	64 GB	8 cores	1 TB NVME SSD (4 TB on Testnet)	100 MBps
Storage Node	Stores data within the 0g network	16 GB	4 cores	500GB / 1T NVME SSD	500 MBps
Storage KV	Handles key-value storage operations	4 GB	2 cores	Matches KV streams size	-
DA Node	Performs blob data verification, signing, and storage	16 GB	8 cores	1 TB NVME SSD	100 MBps
DA Retriever	Retrieves data availability information	8 GB	2 cores	-	100 MBps
DA Encoder*	Encodes data for availability purposes	-	-	-	-
DA Client	Interacts with the Data Availability layer	8 GB	2 cores	-	100 MBps

Note: For DA Encoder, GPU support is currently tested with NVIDIA 12.04 drivers on the RTX 4090. Other NVIDIA GPUs may require parameter adjustments and have not been tuned yet.

Next Steps

Ready to set up your node? Check out our detailed guides:

- Storage Node Setup Guide
- Data Availability Service Setup Guide

Testnet Information

Welcome to Testnet-V3, where you can contribute to our network by operating various node types, including Validator, Storage, and DA (Data Availability) nodes. This page provides an overview of the testnet process and important information for participants.

Add Testnet to Your Wallet

> **Note**
>
> Before adding the 0G-Galileo testnet, please ensure you remove any old testnet configurations from your wallet to avoid conflicts. - Need help?

Choose your preferred wallet provider:

Add to MetaMask

Add to OKX Wallet

0G Testnet Configuration

Summary Table

Detail	Value
Chain Name	0G-Galileo-Testnet
Chain ID	16601
Token Symbol	OG
RPC	https://evmrpc-testnet.0g.ai
Storage Indexer Turbo RPC	https://indexer-storage-testnet-turbo.0g.ai

Detail	Value
Chain Explorer	https://chainscan-galileo.0g.ai/
Storage Explorer	https://storagescan-galileo.0g.ai/
Faucet	https://faucet.0g.ai/

RPCs

3rd Party RPCs (Recommended)

- QuikNode
- ThirdWeb

Contract Addresses

> **CAUTION**
> The contract address might change during the public testnet phase, so please check this page regularly for updates.

Component	Contract	Address
0G Storage Turbo	Flow Contract	`0xbD75117F80b4E22698D0Cd7612d92BDb8eaff628`
	Mine Contract	`0x3A0d1d67497Ad770d6f72e7f4B8F0BAbaa2A649C`
	Market Contract	`0x53191725d260221bBa307D8EeD6e2Be8DD265e19`
	Reward Contract	`0xd3D4D91125D76112AE256327410Dd0414Ee08Cb4`
0G DA	DAEntrance Contract	`0xE75A073dA5bb7b0eC622170Fd268f35E675a957B`

Deployed Block Number: `326165`

Faucet

Faucet

See here for more info.

Explorers

- Chain Scan: Chain Scan provides a comprehensive view of 0G chain activity and transactions.
- Storage Scan: Storage Scan is your go-to tool for exploring storage-related activities within the network.
- Nodes Guru: Nodes Guru provides key information and monitoring tools for validators and node operators to track the health and performance of the network.

See here for more info.

RPC Node

Running a RPC node for the **0G-Galileo-Testnet** in the 0G ecosystem means to provide a public or private RPC service for the community.

Hardware Requirements

Component	Mainnet	Testnet
Memory	64 GB	64 GB
CPU	8 cores	8 cores
Disk	1 TB NVME SSD	4 TB NVME SSD
Bandwidth	100 MBps for Download / Upload	100 MBps for Download / Upload

1. Download Package

Download the latest package for node binaries (named "galileo.tar.gz")

```
wget -O galileo.tar.gz https://github.com/0glabs/0gchain-NG/releases/download/v1.1.1/galileo-v1.1.1.tar.gz
```

2. Extract Package

Unzip this file to your home path

```
tar -xzvf galileo.tar.gz -C ~
```

3. Copy Files and Set Permissions

```
cd galileo
cp -r 0g-home {your data path}
sudo chmod 777 ./bin/geth
sudo chmod 777 ./bin/0gchaind
```

4. Initialize Geth

```
./bin/geth init --datadir /{your data path}/0g-home/geth-home
./genesis.json
```

5. Initialize 0gchaind with Temporary Directory

```
./bin/0gchaind init {node name} --home /{your data path}/tmp
```

6. Copy Node Files to 0gchaind Home

```
cp /{your data path}/tmp/data/priv_validator_state.json /{your data
path}/0g-home/0gchaind-home/data/
cp /{your data path}/tmp/config/node_key.json /{your data path}/0g-
home/0gchaind-home/config/
cp /{your data path}/tmp/config/priv_validator_key.json /{your data
path}/0g-home/0gchaind-home/config/
```

> Note: The temporary directory can be deleted after this step.

7. Start 0gchaind

```
cd ~/galileo
nohup ./bin/0gchaind start \
    --rpc.laddr tcp://0.0.0.0:26657 \
    --chain-spec devnet \
    --kzg.trusted-setup-path=kzg-trusted-setup.json \
    --engine.jwt-secret-path=jwt-secret.hex \
    --kzg.implementation=crate-crypto/go-kzg-4844 \
    --block-store-service.enabled \
    --node-api.enabled \
    --node-api.logging \
    --node-api.address 0.0.0.0:3500 \
    --pruning=nothing \
    --home /{your data path}/0g-home/0gchaind-home \
    --p2p.seeds
```

```
85a9b9a1b7fa0969704db2bc37f7c100855a75d9@8.218.88.60:26656 \
    --p2p.external_address {your node ip}:26656 > /{your data path}/0g-
home/log/0gchaind.log 2>&1 &
```

8. Start Geth

```
cd ~/galileo
nohup ./bin/geth --config geth-config.toml \
         --nat extip:{your node ip} \
         --bootnodes
enode://de7b86d8ac452b1413983049c20eafa2ea0851a3219c2cc12649b971c1677bd83fe
\
         --datadir /{your data path}/0g-home/geth-home \
         --networkid 16601 > /{your data path}/0g-home/log/geth.log 2>&1 &
```

9. Check Chain Status

```
tail -f /{your data path}/0g-home/log/geth.log
tail -f /{your data path}/0g-home/log/0gchaind.log
```

Check logs to confirm your node is running properly.

Important Reminders

- Stay updated with the latest network information and announcements on our socials and blog posts
- Reach out to us on Discord or to the community for support if you encounter any issues

Storage Node

In the 0G network, storage nodes play a vital role in maintaining the system's decentralized storage layer. They are responsible for storing and serving data, ensuring data availability and reliability across the network. By running a storage node, you actively contribute to the network and earn rewards for your participation. This guide details the process of running a storage node, including hardware specifications and interaction with on-chain contracts.

Hardware Requirements

Component	Storage Node	Storage KV
Memory	32 GB RAM	32 GB RAM
CPU	8 cores	8 cores
Disk	500GB / 1TB NVMe SSD	Size matches the KV streams it maintains
Bandwidth	100 Mbps (Download / Upload)	-

NOTE

- For Storage Node: The NVMe SSD ensures fast read/write operations, critical for efficient blob storage and retrieval.
- For Storage KV: The disk size requirement is flexible and should be adjusted based on the volume of KV streams you intend to maintain.

Next Steps

For detailed instructions on setting up and operating your Storage Node or Storage KV, please refer to our comprehensive setup guides below:

Storage Node | **Storage KV Node**

Prerequisites

Before setting up your storage node:

- Understand that 0G Storage interacts with on-chain contracts for blob root confirmation and PoRA mining.
- Check here for deployed contract addresses.

Install Dependencies

Start by installing all the essential tools and libraries required to build the 0G storage node software.

Linux Mac

```
sudo apt-get update
sudo apt-get install clang cmake build-essential pkg-config libssl-dev
```

Install `rustup`: rustup is the Rust toolchain installer, necessary as the 0G node software is written in Rust.

```
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
```

Download the Source Code:

```
git clone -b <latest_tag> https://github.com/0glabs/0g-storage-node.git
```

Build the Source Code

```
cd 0g-storage-node
# Build in release mode
cargo build --release
```

This compiles the Rust code into an executable binary. The `--release` flag optimizes the build for performance.

Configuration

Navigate to the run directory and open config.toml for editing. Follow the steps below.

1. Edit the configuration file:

```
cd run
nano config.toml
```

2. Update configuration with your preferred settings:

Below is just an example configuration for illustration purposes. For official default values, copy over the `config-testnet-turbo.toml` file to your `config.toml` file.

```
# Peer nodes: A list of peer nodes to help your node join the network.
Check inside 0g-storage/run directory for suggested configurations.
network_boot_nodes = []
# Contract addresses
log_contract_address = "CONTRACT_ADDRESS" #flow contract address, see
testnet information
mine_contract_address = "CONTRACT_ADDRESS" #Address of the smart contract
on the host blockchain that manages mining.
# L1 host blockchain RPC endpoint URL. See testnet information page for
RPC endpoints
blockchain_rpc_endpoint = "RPC_ENDPOINT"
# Start sync block number: The block number from which your node should
start synchronizing the log data.
log_sync_start_block_number = BLOCK_NUMBER
# Your private key (64 chars, no '0x' prefix, include leading zeros):
Your private key (without the `0x` prefix) if you want to participate in
PoRA mining and earn rewards.
miner_key = "YOUR_PRIVATE_KEY"
# Max chunk entries in db (affects storage size): The maximum number of
chunk entries (each 256 bytes) to store in the database. This effectively
limits the database size.
db_max_num_chunks = MAX_CHUNKS
# ENR address: Your node's public IP address, essential for other nodes
to discover and connect to you. Currently automatically set by the node.
# network_enr_address = ""
```

Running the Storage Node

1. Check configuration options:

```
../target/release/zgs_node -h
```

2. Run the storage service:

```
cd run
../target/release/zgs_node --config config.toml --miner-key
<your_private_key>
```

Snapshot

Make sure to only include `flow_db` and delete `data_db` under `db` folder when you use a snapshot from a 3rd party !

> Using others' `data_db` will make the node mine for others!

Additional Notes

- **Security:** Keep your private key (`miner_key`) safe and secure. Anyone with access to it can control your node and potentially claim your mining rewards.
- **Network Connectivity:** Ensure your node has a stable internet connection and that the necessary ports are open for communication with other nodes.
- **Monitoring:** Monitor your node's logs and resource usage to ensure it's running smoothly.
- **Updates:** Stay informed about updates to the 0G storage node software and follow the project's documentation for any changes in the setup process.

Remember: Running a storage node is a valuable contribution to the 0G network. You'll be helping to maintain its decentralization and robustness while earning rewards for your efforts.

Data Availability Node

While there are various approaches to running a DA (Data Availability) node, this guide outlines our recommended method and the necessary hardware specifications. DA Nodes perform the core functions of verifying, signing, and storing encoded blob data.

To operate effectively, your DA signer needs to run a DA node to verify encoded blob data, sign it, and store it for future farming and rewards. Currently, to run a DA Node on Testnet, users must stake 10 OG tokens. These can be obtained through our faucet or via rewards from running Storage Nodes or Validator Nodes. You can also reach out to our technical moderators on Discord.

Hardware Requirements

Node Type	Memory	CPU	Disk	Bandwidth	Additional Notes
DA Node	16 GB	8 cores	1 TB NVMe SSD	100 MBps	For Download/Upload

Standing up a DA Node and DA Signer

Run with Docker | **Build from Source** | **Become a Signer**

1. Clone the DA Node Repo:

```
git clone https://github.com/0glabs/0g-da-node.git
cd 0g-da-node
```

2. Generate BLS Private Key (if needed):

If you don't have a BLS private key, generate one:

```
cargo run --bin key-gen
```

Keep the generated BLS private key secure.

3. Set up config.toml:

1. Create a configuration file named `config.toml` in the project root directory.
2. Add the following content to the file, adjusting values as needed:

```
log_level = "info"
data_path = "/data"
# path to downloaded params folder
encoder_params_dir = "/params"
# grpc server listen address
grpc_listen_address = "0.0.0.0:34000"
# chain eth rpc endpoint
eth_rpc_endpoint = "https://evmrpc-testnet.0g.ai"
# public grpc service socket address to register in DA contract
# ip:34000 (keep same port as the grpc listen address)
# or if you have dns, fill your dns
socket_address = "<public_ip/dns>:34000"
# data availability contract to interact with
da_entrance_address = "0x857C0A28A8634614BB2C96039Cf4a20AFF709Aa9" #
testnet config
# deployed block number of da entrance contract
start_block_number = 940000 # testnet config
# signer BLS private key
signer_bls_private_key = ""
# signer eth account private key
signer_eth_private_key = ""
# miner eth account private key, (could be the same as
`signer_eth_private_key`, but not recommended)
miner_eth_private_key = ""
# whether to enable data availability sampling
enable_das = "true"
```

   Make sure to fill in the `signer_bls_private_key`, `signer_eth_private_key`, and `miner_eth_private_key` fields with your actual private keys.

4. Build and Start the Docker Container:

```
docker build -t 0g-da-node .
docker run -d --name 0g-da-node 0g-da-node
```

5. Verify the Node is Running

On the first run, the DA node will register the signer information in the DA contract. You can monitor the console output to ensure the node is running correctly and has successfully registered.

Node Operations

As a DA node operator, your node will perform the following tasks:

- Encoded blob data verification
- Signing of verified data
- Storing blob data for further farming
- Receiving rewards for these operations

Troubleshooting

- If you encounter any issues, check the console output for error messages.
- Ensure that the ports specified in your `config.toml` file are not being used by other applications.
- Verify that you have the latest stable version of Rust installed.
- Make sure your system meets the minimum hardware requirements.

Conclusion

You have now successfully set up and run a 0g DA node as a DA Signer. For more advanced configuration options and usage instructions, please refer to the Official GitHub repository.

Remember to keep your private keys secure and regularly update your node software to ensure optimal performance and security.

Community Docker Repository

This section provides a list of Docker images for 0G DA from the community. For instructions on running 0G nodes via binary installation, please visit the node pages directly.

For most users, Docker offers the simplest method to get 0G nodes up and running. Docker is a platform for containerization, allowing 0G nodes to operate in an isolated environment. This approach enables you to run 0G nodes on your system without needing to install and configure all the necessary dependencies manually.

Most of the officially endorsed 0G Docker implementations can be found under the documentation page for each 0G node type.

Below is a list of community-maintained Docker images for 0G DA. Please note that these images are not officially endorsed by 0G, and users should proceed with caution.

All Node Types

Ember Stake

Validator Node

CryptoWarden

Build with 0G

Start Building

0G Chain

- Deploy Smart Contracts
- Utilize precompiled contracts

0G Compute

- Set up as a Service Provider
- Use the Compute SDK

0G Storage

- Storage SDK Integration
- Storage CLI Usage

0G Data Availability

- DA Integration Guide
- Build Rollups

Community Projects

Explore our growing ecosystem of DeAI applications in the awesome-0g repository, which showcases community projects, tools, and resources built on 0G.

Deploying Contracts on 0G Chain

0G Chain is an EVM-compatible blockchain, with the current version supporting compatibility with Geth 1.10 and the Istanbul upgrade. This means you can generate and deploy contracts just as you would on any EVM chain, taking advantage of the tools and processes you're already familiar with.

Steps to Deploy Your Contract

1. **Prepare Your Smart Contract Code**:
    - Write your contract code as you would for any Ethereum-compatible blockchain, ensuring that it meets the requirements for your specific use case.
2. **Compile Your Smart Contract**:
    - Use `solc` or another compatible Solidity compiler to compile your smart contract.
    - **Important**: When compiling, specify `--evm-version istanbul` to ensure compatibility with the Istanbul upgrade supported by 0G Chain.
    - Example command:

      ```
      solc --evm-version istanbul --bin --abi YourContract.sol
      ```

    - This step will generate the binary and ABI (Application Binary Interface) for your contract.
3. **Deploy the Contract on 0G Chain**:
    - Once compiled, you can use your preferred Ethereum-compatible deployment tools, such as `web3.js`, `ethers.js`, or `hardhat`, to deploy the contract on 0G Chain.
    - Follow the same deployment steps as you would on Ethereum, using your 0G Chain node or RPC endpoint.
4. **Verify Deployment Results on 0g chain scan**:
    - After deployment, you can view the transaction results and verify the contract on 0g chain scan, 0G Chain's block explorer.
    - This explorer will show details about the contract deployment, such as the transaction status, contract address, and interaction history. You can also verify the contract's code and ABI for public transparency.

By following these steps, you’ll be able to deploy and verify your contracts effectively on 0G Chain, leveraging its EVM compatibility for a seamless development experience.

EVM Precompiles on 0G Chain

EVM (Ethereum Virtual Machine) precompile contracts are special, built-in contracts provided by the Ethereum protocol to perform specific, commonly-used operations more efficiently than if they were implemented in Solidity or another high-level language.

The current version of 0g chain supports the Istanbul version of the EVM and all the EVM precompiles it includes(check it out here). In addition to the native EVM precompiles, we have also defined additional precompile contracts to enable modifying the state of Cosmos modules through EVM transactions:

- DASigners precompile
- Staking Precompile
- WrappedOGBase Precompile

Interact with 0G Precompiles in Smart Contracts

Calling 0g precompiles in a contract can be a bit tricky. Since some of the newly added precompiles are stateful, interacting with their non-read-only functions in a contract requires using a low-level call. The `mint` function of WrappedOG can be used as a reference for implementation.

Overview

WrappedOGBase is a wrapper for the `x/wrapped-og-base` module in the 0g chain. WA0GI is a wrapped ERC20 token for native OG. It supports quota-based mint/burn functions based on native OG transfers, on top of traditional wrapped token implementation. The minting/burning quota for each address will be determined through governance voting. `x/wrapped-og-base` is the module that supports and maintains the minting/burning quota.

In most cases this precompile should be only called by WA0GI contract.

Address

`0x0000000000000000000000000000000000001002`

Interface

https://github.com/0glabs/0g-chain/blob/dev/precompiles/interfaces/contracts/IWrappedA0GIBase.sol

Structs

`Supply`

```
struct Supply {
    uint256 cap;
    uint256 initialSupply;
    uint256 supply;
}
```

- **Description**: Defines the supply details of a minter, including the cap, initial supply, and the current supply.
- **Fields**:
  - `cap`: The maximum allowed mint supply for the minter.
  - `initialSupply`: The initial mint supply to the minter, equivalent to the initial allowed burn amount.
  - `supply`: The current mint supply used by the minter, set to `initialSupply` at beginning.

Functions

`getWA0GI()`

```
function getWA0GI() external view returns (address);
```

- **Description**: Retrieves the address of the wrapped OG (WA0GI) contract.
- **Returns**: `address` of the WOG contract.

`minterSupply(address minter)`

```
function minterSupply(address minter) external view returns (Supply
memory);
```

- **Description**: Retrieves the mint supply details for a given minter.
- **Parameters**:
  - `minter`: The address of the minter.
- **Returns**: A `Supply` structure containing the mint cap, initial supply, and current supply of the specified minter.

`mint(address minter, uint256 amount)`

```
function mint(address minter, uint256 amount) external;
```

- **Description**: Mints OG to WA0GI contract and adds the corresponding amount to the minter's mint supply. If the minter's final mint supply exceeds their mint cap, the transaction will revert.
- **Parameters**:
  - `minter`: The address of the minter.
  - `amount`: The amount of OG to mint.
- **Restrictions**: Can only be called by the WA0GI contract.

`burn(address minter, uint256 amount)`

```

```

```
function burn(address minter, uint256 amount) external;
```

- **Description**: Burns the specified amount of OG in WA0GI contract on behalf of the minter and reduces the corresponding amount from the minter's mint supply.
- **Parameters**:
  - `minter`: The address of the minter.
  - `amount`: The amount of OG to burn.
- **Restrictions**: Can only be called by the WOG contract.

Overview

DAsigners is a wrapper for the `x/dasigners` module in the 0g chain, allowing querying the state of this module from EVM calls.

Address

`0x0000000000000000000000000000000000001000`

Interface

https://github.com/0glabs/0g-chain/blob/dev/precompiles/interfaces/contracts/IDASigners.sol

Structs

`SignerDetail`

```
struct SignerDetail {
    address signer;
    string socket;
    BN254.G1Point pkG1;
    BN254.G2Point pkG2;
}
```

- **Description**: Contains details of a signer, including the address, socket, and bn254 public keys (G1 and G2 points).
- **Fields**:
  - `signer`: The address of the signer.
  - `socket`: The socket associated with the signer.
  - `pkG1`: The G1 public key of the signer.
  - `pkG2`: The G2 public key of the signer.

`Params`

```
struct Params {
    uint tokensPerVote;
    uint maxVotesPerSigner;
```

```
    uint maxQuorums;
    uint epochBlocks;
    uint encodedSlices;
}
```

- **Description**: Defines parameters for the DAsigners module.
- **Fields**:
  - `tokensPerVote`: The number of tokens required for one vote.
  - `maxVotesPerSigner`: The maximum number of votes a signer can cast.
  - `maxQuorums`: The maximum number of quorums allowed.
  - `epochBlocks`: The number of blocks in an epoch.
  - `encodedSlices`: The number of encoded slices in one DA blob.

Functions

`params()`

```
function params() external view returns (Params memory);
```

- **Description**: Retrieves the current parameters of the DAsigners module.
- **Returns**: `Params` structure containing the current module parameters.

`epochNumber()`

```
function epochNumber() external view returns (uint);
```

- **Description**: Returns the current epoch number.
- **Returns**: `uint` representing the current epoch number.

`quorumCount(uint _epoch)`

```
function quorumCount(uint _epoch) external view returns (uint);
```

- **Description**: Returns the number of quorums for a given epoch.

- **Parameters**:
  - `_epoch`: The epoch number.
- **Returns**: `uint` representing the quorum count for the given epoch.

`isSigner(address _account)`

```
function isSigner(address _account) external view returns (bool);
```

- **Description**: Checks if a given account is a registered signer.
- **Parameters**:
  - `_account`: The address to check.
- **Returns**: `bool` indicating whether the account is a signer.

`getSigner(address[] memory _account)`

```
function getSigner(
    address[] memory _account
) external view returns (SignerDetail[] memory);
```

- **Description**: Retrieves details for the signers of the provided addresses.
- **Parameters**:
  - `_account`: An array of addresses to fetch the signer details for.
- **Returns**: An array of `SignerDetail` structures for each signer.

`getQuorum(uint _epoch, uint _quorumId)`

```
function getQuorum(
    uint _epoch,
    uint _quorumId
) external view returns (address[] memory);
```

- **Description**: Returns the addresses of the members in a specific quorum for a given epoch.
- **Parameters**:
  - `_epoch`: The epoch number.
  - `_quorumId`: The ID of the quorum.
- **Returns**: An array of addresses that are members of the quorum.

`getQuorumRow(uint _epoch, uint _quorumId, uint32 _rowIndex)`

```
function getQuorumRow(
    uint _epoch,
    uint _quorumId,
    uint32 _rowIndex
) external view returns (address);
```

- **Description**: Retrieves a specific address from a quorum's row for a given epoch and quorum ID.
- **Parameters**:
  - `_epoch`: The epoch number.
  - `_quorumId`: The quorum ID.
  - `_rowIndex`: The row index within the quorum.
- **Returns**: The address at the specified row index in the quorum.

`registerSigner(SignerDetail memory _signer, BN254.G1Point memory _signature)`

```
function registerSigner(
    SignerDetail memory _signer,
    BN254.G1Point memory _signature
) external;
```

- **Description**: Registers a new signer with the provided details and signature.
- **Parameters**:
  - `_signer`: The details of the signer to register.
  - `_signature`: The signature to verify the registration.

`updateSocket(string memory _socket)`

```
function updateSocket(string memory _socket) external;
```

- **Description**: Updates the socket used by the module.
- **Parameters**:
  - `_socket`: The new socket address to update.

`registeredEpoch(address _account, uint _epoch)`

```
function registeredEpoch(
    address _account,
    uint _epoch
) external view returns (bool);
```

- **Description**: Checks if a specific account is registered in a given epoch.
- **Parameters**:
  - `_account`: The address to check.
  - `_epoch`: The epoch number.
- **Returns**: `bool` indicating whether the account is registered for the specified epoch.

`registerNextEpoch(BN254.G1Point memory _signature)`

```
function registerNextEpoch(BN254.G1Point memory _signature) external;
```

- **Description**: Registers the next epoch using the provided signature.
- **Parameters**:
  - `_signature`: The signature used to register the next epoch.

`getAggPkG1(uint _epoch, uint _quorumId, bytes memory _quorumBitmap)`

```
function getAggPkG1(
    uint _epoch,
    uint _quorumId,
    bytes memory _quorumBitmap
) external view returns (BN254.G1Point memory aggPkG1, uint total, uint
hit);
```

- **Description**: Retrieves the aggregated public key for a given epoch and quorum ID.
- **Parameters**:
  - `_epoch`: The epoch number.
  - `_quorumId`: The quorum ID.
  - `_quorumBitmap`: The quorum bitmap.
- **Returns**:

- `aggPkG1`: The aggregated public key.
- `total`: The number of rows.
- `hit`: The number of rows that contributed to the aggregation.

INFTs: Tokenizing AI Agents

The rapid growth of AI agents necessitates new methods for managing their ownership, transfer, and capabilities within Web3 ecosystems. INFTs (Intelligent Non-Fungible Tokens) represent a significant advancement in this space, enabling the tokenization of AI agents to provide transferability, decentralized control, full asset ownership, and royalty potential.

Challenges with Existing NFT Standards

Traditional NFT standards like ERC-721 and ERC-1155 have significant limitations when applied to AI agents, particularly concerning the handling of their unique and often sensitive "intelligence" or metadata:

- **Static and Public Metadata:** Existing standards typically link to static, publicly accessible metadata (e.g., a URI pointing to a JSON file). AI agents, however, require dynamic metadata that reflects their learning and evolution, and this data often needs privacy protection.
- **Insecure Metadata Transfer:** When transferring an ERC-721 token, only the ownership identifier moves—not the underlying metadata. For AI agents, this means the new owner might receive an incomplete or non-functional agent.
- **Lack of Native Encryption/Privacy:** Current standards don't inherently support the encryption needed to protect proprietary AI models or sensitive user data contained within an agent's metadata.

ERC-7857: A Standard Designed for AI Agents

ERC-7857 is a new NFT standard introduced by 0G Labs, specifically designed to address these shortcomings. It enables the creation, ownership, and secure transfer of INFTs with their complete intelligence intact.

Key Features & Advantages

- **Privacy-Preserving Metadata:** Allows sensitive metadata (the agent's "intelligence") to be encrypted and kept private, protecting proprietary information.
- **Secure Metadata Transfers:** Ensures that when an INFT is transferred, both the ownership and the encrypted metadata are securely passed to the new owner in a verifiable way.

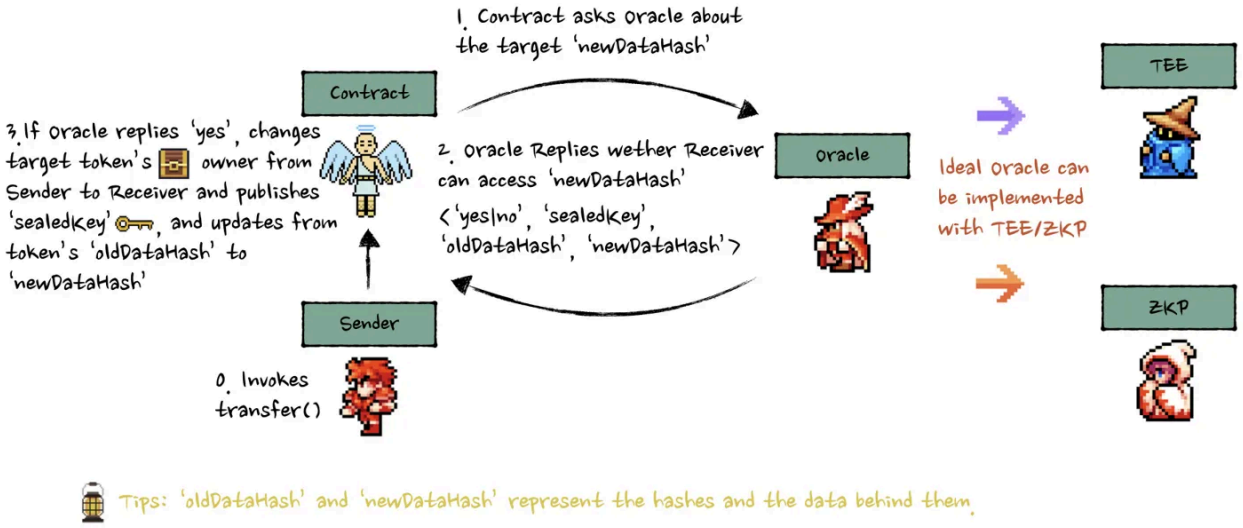
- **Dynamic Data Management:** Supports the dynamic nature of AI agents, allowing their metadata (state, models, behaviors) to be updated securely within the NFT framework.
- **Decentralized Storage Integration:** Works with systems like 0G Storage for secure, permanent, and tamper-proof storage and access management of metadata.
- **Verifiable Ownership & Control:** Uses cryptographic proofs and oracles to validate metadata transfers, ensuring integrity.
- **AI-Specific Functionality:** Built for AI use cases, enabling features like agent lifecycle management and ownership verification before task execution.

How ERC-7857 Works

The transfer mechanism in ERC-7857 is designed to ensure that both token ownership and access to the token's encrypted metadata are securely transferred. This process involves several key components:

Core Process Flow

1. **Encryption & Commitment:** The AI agent's metadata is encrypted. A hash (commitment) of this encrypted data is created as proof of authenticity without revealing the content.
2. **Secure Transfer Initiation:** When the INFT is transferred, a trusted oracle (potentially using secure environments like TEEs) decrypts the original metadata.
3. **Re-encryption for Receiver:** The oracle generates a new encryption key, re-encrypts the metadata with it, and stores this new encrypted metadata (e.g., on 0G Storage).
4. **Key Delivery:** The new encryption key is encrypted using the receiver's public key. This ensures only the intended new owner can access the actual metadata key.
5. **Verification & Finalization:** The transfer function on the smart contract verifies proofs: the sender's access, the oracle's validation that the new metadata matches the old, and the receiver's signed acknowledgment. If valid, the NFT ownership transfers, and the receiver gets the encrypted metadata key.
6. **Access Granted:** The receiver uses their private key to decrypt the metadata key, granting them full access to the agent's encrypted intelligence.



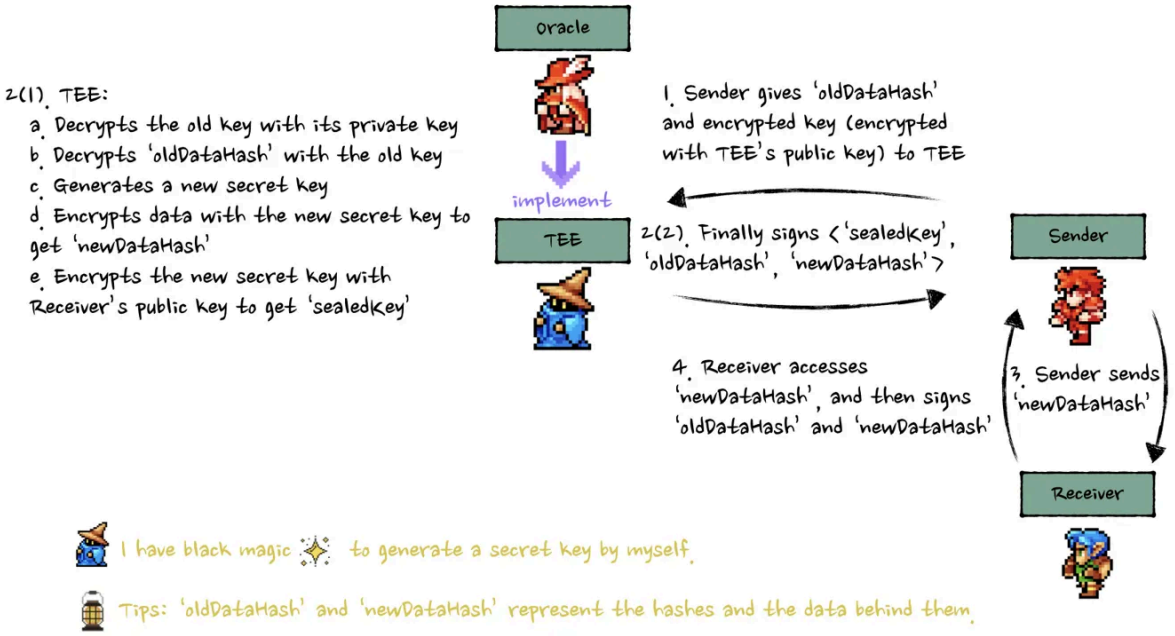
Oracle Implementations

ERC-7857 supports two primary oracle implementations:

TEE (Trusted Execution Environment)

In the TEE implementation:

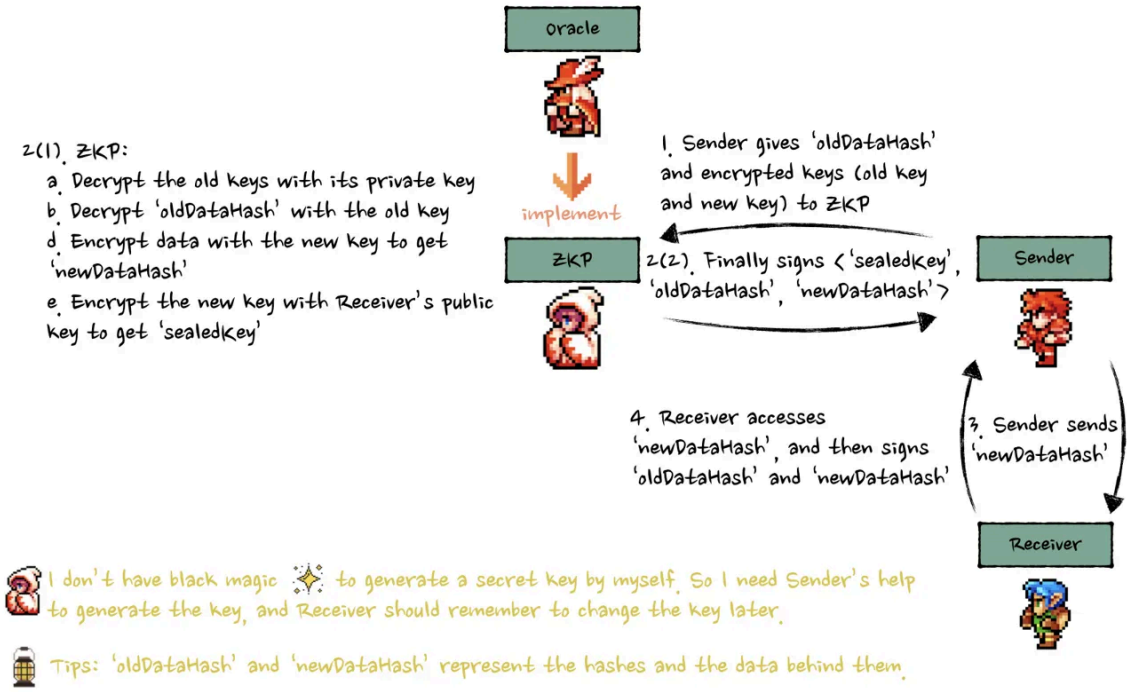
- The sender transmits the encrypted data and key to the TEE
- The TEE securely decrypts the data, generates a new key, and re-encrypts the metadata
- The TEE encrypts the new key with the receiver's public key
- The TEE outputs the sealed key and corresponding hash values
- This approach offers strong security as the TEE can generate its own secure keys



ZKP (Zero-Knowledge Proof)

In the ZKP implementation:

- The sender provides both old and new keys to the ZKP system
- The ZKP verifies that the metadata was correctly re-encrypted
- Unlike TEE, ZKP cannot independently generate new keys
- Receivers should change keys after transfer for enhanced security



Additional Functions

ERC-7857 also supports:

- **clone():** Similar to transfer() but creates a new token with the same metadata instead of changing ownership of the original token.
- **authorizeUsage():** Adds authority for using the token's private metadata without granting access to the raw data, requiring a sealed executor that processes the metadata securely.

Applications and Use Cases

The ability to securely tokenize and transfer AI agents opens up numerous possibilities:

- **AI Agent Marketplaces:** Platforms for buying and selling trained AI agents, with guaranteed secure transfer of their capabilities.
- **Personalized Automation:** Owning and trading INFTs tailored for specific tasks (e.g., DeFi operations, airdrop claiming).
- **Enterprise AI Solutions:** Building, owning, and securely transferring or leasing proprietary AI agents for internal or client use.
- **AI-as-a-Service (AIaaS):** Tokenizing and leasing AI agents on subscription models.
- **Agent Collaboration:** Combining or composing different INFT agents to create more powerful tools.
- **Intellectual Property Monetization:** Allowing AI developers to monetize their models as NFTs while controlling usage.

Integration with 0G Infrastructure

INFTs leverage the 0G ecosystem in several key ways:

- **0G Storage:** Provides the secure, decentralized storage layer required for encrypted metadata, ensuring it remains accessible only to legitimate owners.
- **0G DA (Data Availability):** Guarantees the verified availability of metadata proofs necessary for the transfer verification process.
- **0G Chain:** Enables fast, scalable execution of INFT operations at lower cost than existing solutions.
- **0G Compute Network:** Can be utilized by INFTs for performing secure inferences, enabling AI agents to execute tasks without exposing their underlying model.

By combining INFTs with 0G's comprehensive AI infrastructure, developers can create sophisticated, transferable AI agents that maintain their intelligence, privacy, and

functionality throughout their lifecycle.

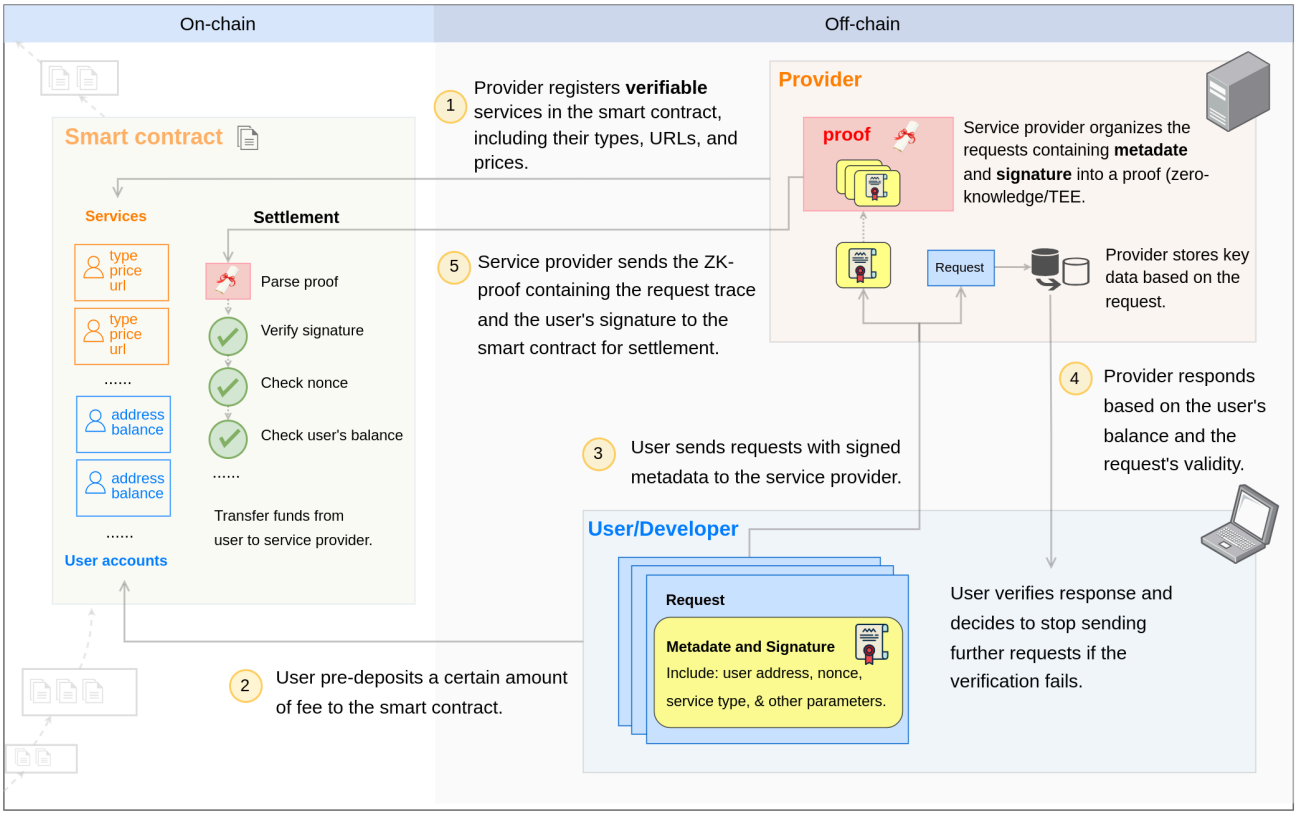
Getting Started

To begin working with INFTs on the 0G ecosystem, developers can reference our GitHub repository for sample implementations and integration examples. The ERC-7857 standard is designed to be compatible with existing Web3 infrastructure while providing the enhanced security and functionality needed for AI agent tokenization.

Overview

The 0G Compute Network connects AI users and AI service providers, making it easy for AI users to access a wide range of compute and model services. As part of this, the framework is built to provide permissionless settlement between AI users and AI service providers in a fast and trustworthy manner, as required by a fully distributed AI economy.

We integrate various stages of the AI process to make these services both verifiable and billable. This ensures that Service Providers, such as platforms or users offering compute resources, can deliver trusted and accountable solutions.



Components

Contract: This component determines the legitimacy of settlement proofs, manages accounts, and handles service information. It stores variables during the service process, including account information, service details (such as name and URL), and consensus logic.

Provider: The owners of AI models and hardware who offer their services for a fee.

User: Individuals or organizations who use the services listed by Service Providers. They may use AI services directly or build applications on top of our API.

Process Overview

The 0G Compute Network implements the following workflow:

1. **Service Registration:** Providers register their services' types, URLs, and prices in the smart contract.
2. **Fee Staking:** Users deposit a certain amount into the smart contract for service fees. If the accumulated charges from user requests exceed their deposit, the provider will stop responding.
3. **Request Submission:** Users or developers send requests, along with metadata and signatures, to the Service Provider.
4. **Provider Response:** Providers respond based on the user's balance and the request's validity.
5. **Settlement and Verification:** Providers generate a proof (zero-knowledge proof (ZK-proof)/[TEE proof](#SOME TAG)) and submit it to the smart contract for verification and settlement.
6. **User Verification:** Users verify the Service Provider's response and can stop requests if the verification fails.

This brief overview introduces the foundational workflow. For more detailed steps, please refer to the full documentation.

Get Involved

The 0G Compute Network supports two main types of services: Inference and Fine-Tuning. Each service offers unique opportunities for providers to contribute to the AI ecosystem and additional features planned for future releases.

To become an **Inference Service Provider**, please refer to the Inference Provider section for detailed guidelines and requirements.

To become a **Fine-Tuning Service Provider**, please refer to the Fine-Tuning Provider section for comprehensive instructions and criteria.

If you wish to leverage provider services to develop your own projects, relevant resources are available in the Developer SDK section for inference.

If you wish to use your own data to optimize existing models, please refer to the Fine-Tuning CLI section to utilize the services provided by the compute network.

For those looking to use the 0G Compute Network to access AI services, more information can be found in the Marketplace section.

Inference Provider

To integrate your AI services into the 0G Compute Network and become a Service Provider, you must first transform your service into a verifiable service and connect it through the provider broker container.

This guide will walk you through the process of setting up your verifiable model service and integrating it with the network.

Verifiable Services

Service Interface Requirements

Large Language Models (LLMs) revolutionize communication, knowledge access, and automation by generating human-like text, so we start with supporting language models. To have a consistent experience, providers shall support the OpenAI API Interface Standards.

Verification Interfaces

To ensure the integrity and trustworthiness of services, different verification mechanisms are employed. Each mechanism comes with its own specific set of protocols and requirements to ensure service verification and security.

TEEML | **OPML, ZKML, and others**

For TEE (Trusted Execution Environment) verification, when a service starts, it should generate a signing key within the TEE. We require CPU and GPU attestations to ensure the service is running in a Confidential VM with an NVIDIA GPU in TEE mode. These attestations should include the public key of the signing key, verifying its creation within the TEE. All inference results must be signed with this signing key.

Note: Ensure that Intel TDX (Trusted Domain Extensions) is enabled on the CPU. Additionally, an H100 or H200 GPU is required for GPU TEE.

1. Attestation Download Interface

To facilitate attestation downloads, set up an API endpoint at:

```
GET https://{PUBLIC_IP}/attestation/report
```

This endpoint should return a JSON structure in the following format:

```
{
  "signing_address": "...",
  "nvidia_payload": "..."
}
```

Note: Ensure that the "nvidia_payload" can be verified using NVIDIA's GPU Attestation API. Support for decentralized TEE attestation is planned for the future, and relevant interfaces will be provided. Stay tuned.

2. Signature Download Interface

To facilitate the downloading of response signatures, provide an API endpoint at:

```
GET https://{PUBLIC_IP}/signature/{response_id}
```

Each response should include a unique ID that can be utilized to retrieve its signature using the above endpoint.

- **Signature Generation**: Ensure the signature is generated using the ECDSA algorithm.
- **Verification**: The signature should be verifiable with the signing address, along with the corresponding request and response content.

Provider Broker

To register and manage services, handle user request proxies, and perform settlements, you need to use the Provider Broker.

Prerequisites

- Docker Compose: 1.27+

Download the Installation Package

Please visit the releases page to download and extract the latest version of the installation package.

Configuration Setup

- Copy the `config.example.yaml` file.

- Modify `servingUrl` to point to your publicly exposed URL.
- Set `privateKeys` to your wallet's private key for the 0G blockchain.
- Set `servingUrl` to your service's public URL.
- Set `targetUrl` to your internal URL corresponding to the prepared LLM service.
- Set `model` to the model name of your LLM service.
- Save the file as `config.local.yaml`.
- Replace `#PORT#` in `docker-compose.yml` with the port you want to use. It should be the same as the port of `servingUrl` in `config.local.yaml`.

Start the Provider Broker

```
docker compose -f docker-compose.yml up -d
```

The provider broker has an automatic settlement engine that ensures you can collect fees promptly before your customer's account balance is insufficient, while also minimizing the frequency of charges to reduce gas consumption.

Inference SDK

Overview

The 0G Compute Network SDK enables developers to integrate AI inference services from the 0G Compute Network into their applications. Currently, the 0G Compute Network SDK supports Large Language Model (LLM) inference services, with fine-tuning and additional features planned for future releases.

In just five minutes, you can initialize your broker to manage operations, set up and fund your account to pay for inference services, and learn how to send inference requests and handle responses.

Features

- Easy integration with AI providers on the 0G Network with built-in billing and account management
- Compatible with OpenAI SDK format
- Support for verifiable responses

TypeScript SDK | **Other SDKs**

Installation

```
pnpm add @0glabs/0g-serving-broker @types/crypto-js@4.2.2 crypto-js@4.2.0
```

Quick Start Guide

Initialize the Broker

The broker is your main interface to the 0G Compute Network. It handles authentication, billing, and service communication.

Before initializing the broker, you need:

1. A wallet signer (implements either `JsonRpcSigner` or `Wallet` from ethers package)
2. (Optional) The contract address for the 0G Serving contract

Option 1: Using a private key

```
const { ethers } = require("ethers");
const { createZGComputeNetworkBroker } = require("@0glabs/0g-serving-broker");
const provider = new ethers.JsonRpcProvider("https://evmrpc-testnet.0g.ai");
// Get the signer
const privateKey = "INPUT_PRIVATE_KEY_HERE";
const wallet = new ethers.Wallet(privateKey, provider);
// Initialize broker
const broker = await createZGComputeNetworkBroker(wallet);
```

Option 2: Using a browser wallet

```
import { BrowserProvider } from "ethers";
import { createZGComputeNetworkBroker } from "@0glabs/0g-serving-broker";
async function initializeBrokerWithWallet() {
  // Ensure wallet is installed
  if (typeof window.ethereum !== "undefined") {
    // Create a provider
    const provider = new BrowserProvider(window.ethereum);
    // Get the signer
    const signer = await provider.getSigner();
    // Initialize broker
    const broker = await createZGComputeNetworkBroker(signer);
  }
}
```

> The `contractAddress` parameter is optional - the SDK uses a default address if not provided.

Discover Available Services

The 0G Compute Network hosts multiple AI service providers. The service discovery process helps you find and select the appropriate services for your needs.

```
const services = await broker.inference.listService();
```

Each service contains the following information:

```
type ServiceStructOutput = {
  provider: string; // Provider's wallet address (unique identifier)
```

```
  serviceType: string; // Type of service
  url: string; // Service URL
  inputPrice: bigint; // Price for input processing
  outputPrice: bigint; // Price for output generation
  updatedAt: bigint; // Last update timestamp
  model: string; // Model identifier
  verifiability: string; // Indicates how the service's outputs can be verified. 'TeeML' means it runs with verification in a Trusted Execution Environment. An empty value means no verification.
};
```

Currently, we provide two official services:

1. **llama-3.3-70b-instruct**

- Provider address: `0xf07240Efa67755B5311bc75784a061eDB47165Dd`

2. **deepseek-r1-70b**

- Provider address: `0x3feE5a4dd5FDb8a32dDA97Bed899830605dBD9D3`

Account Management

The 0G Compute Network uses a prepaid account system for each provider. Before using any services, you need to set up and fund an account for each provider you want to use.

Create an Account

```
await broker.ledger.addLedger(initialBalance);
```

The `initialBalance` needs to be specified in OG units.

Add Funds

```
await broker.ledger.depositFund(amount);
```

The `amount` needs to be specified in OG units.

Making Service Requests

Service usage in the 0G Network involves two key steps:

1. Retrieving service metadata

2. Generating authenticated request headers

```
// Get service metadata
const { endpoint, model } = await broker.inference.getServiceMetadata(
  providerAddress
);
// Generate request headers
const headers = await broker.inference.getRequestHeaders(
  providerAddress,
  content
);
```

Note: Headers generated by `getRequestHeaders` are single-use only & Each request requires new headers

Send Request

You can make requests using either the native fetch API or the OpenAI SDK.

Using fetch:

```
const response = await fetch(`${endpoint}/chat/completions`, {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
    ...headers,
  },
  body: JSON.stringify({
    messages: [{ role: "system", content }],
    model: model,
  }),
});
```

Using OpenAI SDK:

```
const openai = new OpenAI({
  baseURL: endpoint,
  apiKey: "", // Leave empty
});
const completion = await openai.chat.completions.create(
  {
    messages: [{ role: "system", content }],
    model: model,
```

```
  },
  {
    headers: {
      ...headers,
    },
  }
);
```

Response Processing

This function is used to verify the response and settle the fee. If it is a verifiable service, it will return whether the response is valid.

For streaming services, `content` should be the total content received from the service. And chatID can be any chat ID of the chunk.

```
const valid = await broker.inference.processResponse(
  providerAddress,
  content,
  chatID // Optional: Only for verifiable services
);
```

Manual Fee Settlement

If `processResponse` fails, you can settle fees manually by calling `settleFee` function.

```
await broker.inference.settleFee(providerAddress, fee);
```

Helper Functions

Get Account Details

This function is used to get account details (like balance) on the given provider address.

```
const accountDetails = await broker.ledger.getLedger();
```

Request Refund

This function is used to request a refund from your account.

```
await broker.ledger.retrieveFund("inference", amount);
```

Fine-tuning Provider

This guide provides a comprehensive walkthrough for setting up and offering computing power as a fine-tuning provider on the 0G Compute Network.

Preparation

Download the Installation Package

- **Visit the Releases Page:** 0G Serving Broker Releases
- **Download and Extract:** Get the latest version of the fine-tuning installation package.

Configuration Setup

- **Copy the Config File:** Duplicate `config.example.yaml` to create `config.local.yaml`.

- **Modify Settings:**

  - Set `servingUrl` to your publicly accessible URL.
  - Set `privateKeys` using your wallet's private key for the 0G blockchain.
- **Edit `docker-compose.yml`:** Replace `#PORT#` with the desired port, matching the port in `config.local.yaml`.

- **Supporting Custom Models from Providers**

  To include custom models, refer to the example configuration below and update your `config.local.yaml` file accordingly. Ensure that all required fields are properly defined to match your specific model setup.

```
  service:
    customizedModels:
      - name: "deepseek-r1-distill-qwen-1.5b"
        hash: "<MODEL_ROOT_HASH>"
        image: "deepseek:latest"
        dataType: "text"
        trainingScript: "/app/finetune.py"
        description: "DeepSeek-R1-Zero, a model trained via large-
scale reinforcement learning (RL) without supervised fine-tuning
(SFT) as a preliminary step, demonstrated remarkable performance on
reasoning."
        tokenizer: "<TOKENIZER_ROOT_HASH>"
```

```
        usageFile: "<ZIP_FILE>"
      - name: "mobilenet_v2"
        hash: "<MODEL_ROOT_HASH>"
        image: "mobilenetV2:latest"
        dataType: "image"
        trainingScript: "/app/finetune.py"
        description: "MobileNet V2 model pre-trained on ImageNet-1k
at resolution 224x224."
        tokenizer: "<TOKENIZER_ROOT_HASH>"
        usageFile: "<ZIP_FILE>"
```

Configuration Fields:

- **name:** Model identifier
- **hash:** The root hash of the pre-trained model, obtained after uploading the model to 0G storage.
- **image:** The Docker image that encapsulates the fine-tuning execution environment.
- **dataType:** Specifies the type of dataset the model is intended to train on. Valid options include `text` or `image`.
- **trainingScript:** Specifies the path to the training script within the container. Fine-tuning will be executed using the command `python <trainingScript>`.
- **description:** A concise overview of the model, highlighting its key features and capabilities.
- **tokenizer:** The root hash of the tokenizer files used for dataset processing. This value is obtained after uploading the tokenizer files to 0G storage.
- **usageFile:** The ZIP file (referenced by its name, not the full path) contains detailed usage information for this model, including training configuration examples, build specifications, or sample datasets. Make sure the file is placed in the `./models` directory.

Build the TDX Guest Image

Prerequisites Installation

- **Install Docker:**

```
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
```

- **Add User to Docker Group:**

```
sudo usermod -aG docker $USER
```

```
newgrp docker
```

- **Verify Installation:**

```
docker --version
docker run hello-world
```

Build CVM Image

To ensure secure and private execution of fine-tuning tasks, you will build an image suitable for running in a Confidential Virtual Machine (CVM). This process leverages NVIDIA's TEE GPU technology and Intel CPUs with TDX support, enhancing security by running model training in an isolated environment.

- **Clone Repository:**

```
git clone https://github.com/nearai/private-ml-sdk --recursive
cd private-ml-sdk/
./build.sh
```

- **Image Files Location:** Check out `private-ml-sdk/images/`. Available images include:
  - `dstack-nvidia-0.3.0`: Production image without developer tools.
  - `dstack-nvidia-dev-0.3.0`: Development image with tools like `sshd`, `strace`.

Run Application

Run the Local KMS

The Local KMS provides essential keys for CVM initialization, derived from local TEE hardware.

- **Launch KMS:**

```
cd private-ml-sdk/meta-dstack-nvidia/dstack/key-provider-build/
./run.sh
```

Run the TDX Guest Image

Ensure you have a TDX host machine with the TDX driver and a compatible NVIDIA GPU.

- **Update PATH:**

```
pushd private-ml-sdk/meta-dstack-nvidia/scripts/bin
PATH=$PATH:`pwd`
popd
```

- **List Available GPUs:**

```
dstack lsgpu
```

- **Create CVM Instance:**

  Replace `#PORT#` with your configured port:

```
dstack new docker-compose.yaml -o my-gpu-cvm \
       --local-key-provider \
       --gpu [GPU_ID] \
       --image images/dstack-nvidia-0.3.0 \
       -c 2 -m 4G -d 100G \
       --port tcp:0.0.0.0:#PORT#:#PORT#
```

Run the CVM

- **Copy Config File:**

```
cp config.local.yaml private-ml-sdk/my-gpu-
cvm/shared/config.local.yaml
```

- **Start the CVM:**

```
sudo -E dstack run my-gpu-cvm
```

By following these steps, you will successfully set up your service as a fine-tuning provider on the 0G Compute Network, leveraging secure and verifiable computing environments.

Fine-tuning CLI

Introduction

Overview

The fine-tuning service is an AI service based on the 0G Compute Network that allows users to access computing resources from AI service providers via the 0G Compute Network. Fine-tuning service providers can offer computing resources through the 0G Compute Network to assist AI users in model fine-tuning. This document describes how to interact with the 0G Compute Network using the Fine-tuning CLI.

Setup

Prerequisites

Node version >= 22.0.0

Installation

```
pnpm install @0glabs/0g-serving-broker -g
```

Environment Variables

Before using the Fine-tuning CLI, you need to set the following environment variables: `RPC_ENDPOINT` and `ZG_PRIVATE_KEY`. `RPC_ENDPOINT` is the RPC endpoint of the 0G Compute Network, and `ZG_PRIVATE_KEY` is your private key.

```
export RPC_ENDPOINT=<YOUR_RPC_ENDPOINT>   # default: https://evmrpc-
testnet.0g.ai
export ZG_PRIVATE_KEY=<YOUR_PRIVATE_KEY>
```

Basic Workflow

Create Account

The Fine-tuning CLI requires an account to pay for service fees via the 0G Compute Network. You can create an account with the following command:

```
0g-compute-cli add-account --amount <AMOUNT>
```

List Providers

```
0g-compute-cli list-providers
```

The output will be like:

```
┌──────────────────────────────────────────────────┬──────────────────────────────────────────────────┐
│ Provider 1                                       │ 0xf07240Efa67755B5311b
├──────────────────────────────────────────────────┼──────────────────────────────────────────────────┤
│ Available                                        │ ✓
├──────────────────────────────────────────────────┼──────────────────────────────────────────────────┤
│ Price Per Byte in Dataset (OG)                   │ 0.000000000000000001
├──────────────────────────────────────────────────┼──────────────────────────────────────────────────┤
│ Provider 2                                       │ ......
├──────────────────────────────────────────────────┼──────────────────────────────────────────────────┤
│ ......                                           │ ......
└──────────────────────────────────────────────────┴──────────────────────────────────────────────────┘
```

- **Provider x:** The address of the provider. The address of the official provider is `0xf07240Efa67755B5311bc75784a061eDB47165Dd`.
- **Available:** Indicates if the provider is available. If `✓`, the provider is available. If `✗`, the provider is occupied.
- **Price Per Byte in Dataset (OG):** The service fee charged by the provider. The fee is currently based on the byte count of the dataset. Future versions may charge more accurately based on the token count of the dataset.

List Preset Models

```
0g-compute-cli list-models
```

The output will be like:

```
Predefined Model:
┌──────────────────────────────┬──────────────────────────────────────────────────┐
│ Name                         │ Description
├──────────────────────────────┼──────────────────────────────────────────────────┤
│ distilbert-base-uncased      │ DistilBERT is a transformers model, smalle
│                              │ as pretrained on the same corpus in a self
```

```
│                              │ BERT base model as a teacher. More details
│                              │ ngface.co/distilbert/distilbert-base-uncas
├──────────────────────────────┼──────────────────────────────────────────────
│ cocktailsgd-opt-1.3b         │ CocktailSGD-opt-1.3B finetunes the Opt-1.3
│                              │ GD, which is a novel distributed finetunin
│                              │ e found at: https://github.com/DS3Lab/Cock
└──────────────────────────────┴──────────────────────────────────────────────
Provider's Model:
┌──────────────────────────────┬──────────────────────────────────────────────
│ Name                         │ Description
├──────────────────────────────┼──────────────────────────────────────────────
│ deepseek-r1-distill-qwen-1.5 │ DeepSeek-R1-Zero, a model trained via larg
│ b                            │ (RL) without supervised fine-tuning (SFT)
│                              │ ated remarkable performance on reasoning.
├──────────────────────────────┼──────────────────────────────────────────────
│ mobilenet_v2                 │ MobileNet V2 model pre-trained on ImageNet
└──────────────────────────────┴──────────────────────────────────────────────
```

The output consists of two main sections:

- **Predefined Models**

These are models that are provided by the system as predefined options. They are typically built-in, curated, and maintained to ensure quality, reliability, and broad applicability across common use cases.

- **Provider's Model**

These models are offered by external service providers. Providers may customize or fine-tune models to address specific needs, industries, or advanced use cases. The availability and quality of these models may vary depending on the provider.

Note: We currently offer the models listed above as presets. You can choose one of these models for fine-tuning. More models will be provided in future versions.

Prepare Configuration File

Please download the parameter file template for the model you wish to fine-tune from the releases page and modify it according to your needs.

Note: For custom models provided by third-party Providers, you can download the usage template `including instructions on how to construct the dataset and training configuration` using the following command:

```
```

```
0g-compute-cli model-usage --provider <PROVIDER_ADDRESS>  --model
<MODEL_NAME>   --output <PATH_TO_SAVE_MODEL_USAGE>
```

Prepare Dataset

Please download the dataset format specification and verification script from the releases page to make sure your generated dataset complies with the requirements.

After preparing the dataset, upload it to 0G Storage using the following command:

```
0g-compute-cli upload --data-path <PATH_TO_DATASET>
```

Note: Record the root hash of the dataset; they will be needed in later steps.

Calculate Dataset Size

After uploading the dataset to storage, you can calculate its size by running the following command:

```
0g-compute-cli calculate-token --model <MODEL_NAME> --dataset-path
<PATH_TO_DATASET> --provider <PROVIDER_ADDRESS>
```

- **--model:** The name of the model you intend to use; see List Preset Models
- **--dataset-path:** The local path to the dataset that you want to evaluate.
- **--provider (option):** Address of the service provider. This is only required when you are using a **Provider's Model** instead of a **Predefined Model**.

Create Fine-Tuning Task

Once you've chosen a pretrained model and prepared your dataset and configuration file, you can create a fine-tuning task with the following command:

```
0g-compute-cli  create-task  --provider <PROVIDER_ADDRESS> --model
<MODEL_NAME> --dataset <DATASET_ROOT_HASH> --config-path
<CONFIG_FILE_PATH> --data-size <DATA_SIZE> --gas-price <GAS_PRICE>
```

- **--provider:** Address of the service provider; see List Providers
- **--model:** Name of the pretrained model; see List Preset Models
- **--dataset:** Root hash of the dataset; see Prepare Dataset
- **--config-path:** Path to the parameter file; see Prepare Configuration File

- **--data-size:** Size of the dataset; see Calculate Dataset Size
- **--gas-price:** Gas price. If not specified, a default value calculated by the client will be used.

The output will be like:

```
Verify provider...
Provider verified
Creating task...
Created Task ID: 6b607314-88b0-4fef-91e7-43227a54de57
```

Note: When creating a task for the same provider, you must wait for the previous task to be completed (status `Finished`) before creating a new task. If the provider is currently running other tasks, you will be prompted to choose between adding your task to the waiting queue or canceling the request.

Check Task

You can check the status of a task using the command below:

```
0g-compute-cli get-task --provider <PROVIDER_ADDRESS> --task <TASK_ID>
```

- **--provider:** Address of the service provider
- **--task:** Task ID. If not specified, the most recently created task will be displayed.

The output will be like:

```
┌──────────────────────────┬──────────────────────────────────────────────────┐
│ Field                    │ Value                                            │
├──────────────────────────┼──────────────────────────────────────────────────┤
│ ID                       │ beb6f0d8-4660-4c62-988d-00246ce913d2             │
├──────────────────────────┼──────────────────────────────────────────────────┤
│ Created At               │ 2025-03-11T01:20:07.644Z                         │
├──────────────────────────┼──────────────────────────────────────────────────┤
│ Pre-trained Model Hash   │ 0xcb42b5ca9e998c82dd239ef2d20d22a4ae1            │
├──────────────────────────┼──────────────────────────────────────────────────┤
│ Dataset Hash             │ 0xaae9b4e031e06f84b20f10ec629f36c5771            │
├──────────────────────────┼──────────────────────────────────────────────────┤
│ Training Params          │ {......}                                         │
├──────────────────────────┼──────────────────────────────────────────────────┤
│ Fee (neuron)             │ 179668154                                        │
├──────────────────────────┼──────────────────────────────────────────────────┤
```

```
| Progress                                | Delivered
└─────────────────────────────────────────┴──────────────────────────────────────────
```

- **ID:** Task ID
- **Pre-trained Model Hash:** Root hash corresponding to the pretrained model stored on storage
- **Dataset Hash:** Root hash corresponding to the dataset stored on storage
- **Training Params:** Parameters used for fine-tuning
- **Fee (neuron):** Task fee.
- **Progress:** Task status. Possible values are `InProgress`, `Delivered`, `UserAckDelivered`, `Finished`, `Failed`. These represent "task in progress", "provider has uploaded the fine-tuning result", "user has confirmed the result is downloadable", "task completed", "task failed" respectively.

View Task Logs

You can view task logs with the following command:

```
0g-compute-cli get-log --provider <PROVIDER_ADDRESS> --task <TASK_ID>
```

Possible output:

```
creating task....
Step: 0, Logs: {'loss': ..., 'accuracy': ...}
...
Training model for task beb6f0d8-4660-4c62-988d-00246ce913d2 completed
successfully
```

Confirm Task Result

Use the Check Task command to view task status. When the status changes to `Delivered`, it indicates that the provider has completed the fine-tuning task and uploaded the result to storage. The corresponding root hash has also been saved to the contract. You can download the model with the following command; CLI will download the model based on the root hash submitted by the provider. If the download is successful, CLI updates the contract information to confirm the model is downloaded.

```
0g-compute-cli acknowledge-model --provider <PROVIDER_ADDRESS>  --data-
path <PATH_TO_SAVE_MODEL>
```

Note: The model file downloaded with the above command is encrypted, and additional steps are required for decryption.

Decrypt Model

The provider will check the contract to verify if the user has confirmed the download, enabling the provider to settle fees successfully on the contract subsequently. Once the provider confirms the download, it uploads the key required for decryption to the contract, encrypted with the user's public key, and collects the fee. You can again use the `get-task` command to view the task status. When the status changes to `Finished`, it means the provider has uploaded the key. At this point, you can decrypt the model with the following command:

```
0g-compute-cli decrypt-model --provider <PROVIDER_ADDRESS> --encrypted-
model <PATH_TO_ENCRYPTED_MODEL> --output <PATH_TO_SAVE_DECRYPTED_MODEL>
```

The above command performs the following operations:

1. Gets the encrypted key from the contract uploaded by the provider
2. Decrypts the key using the user's private key
3. Decrypts the model with the decrypted key

Note: The decrypted result will be saved as a zip file. Ensure that the `<PATH_TO_SAVE_DECRYPTED_MODEL>` ends with `.zip` (e.g., `model_output.zip`). After downloading, unzip the file to access the decrypted model.

Account Management

View Account

```
0g-compute-cli get-account
```

Possible output:

```
  Overview
┌──────────────────────────────────────────┬──────────────────────┐
│ Balance                                  │ Value (OG)           │
├──────────────────────────────────────────┼──────────────────────┤
│ Total                                    │ 0.999999999820331942 │
├──────────────────────────────────────────┼──────────────────────┤
│ Locked (transferred to sub-accounts)     │ 0.000000000179668154 │
└──────────────────────────────────────────┴──────────────────────┘
```

```
  Fine-tuning sub-accounts (Dynamically Created per Used Provider)
┌──────────────────────────────────────────────┬──────────────────────┐
│ Provider                                     │ Balance (OG)         │
├──────────────────────────────────────────────┼──────────────────────┤
│ 0x3C44CdDdB6a900fa2b585dd299e03d12FA4293BC   │ 0.000000000179668154 │
├──────────────────────────────────────────────┼──────────────────────┤
│ ......                                       │ ......               │
└──────────────────────────────────────────────┴──────────────────────┘
```

Overview: Provides a general overview of the account's balance.

- **Total:** The current balance of the account
- **Locked:** The cumulative amount locked in all sub-accounts

Fine-tuning sub-accounts: Information about sub-accounts, with each sub-account corresponding to a provider for paying the provider's service fee. Each sub-account is dynamically created when tasks are submitted.

- **Provider:** Address of the provider corresponding to the sub-account
- **Balance:** Balance of the sub-account, which is an amount transferred from the main account to the sub-account based on the task fee whenever a task is created.
- **Requested Return to Main Account:** Amount requested to be returned from sub-accounts to the main account. If the amount in the sub-account goes unspent for any reason, such as a task failure, you can use the `return-funds` command to return the balance to the main account. However, it won't return immediately and will only be available after a lock-in period. For details, refer to Retrieving Funds.

Note: For more information about sub-accounts, refer to View Sub-Account.

Deposit

You can deposit into your account using the following command.

```
0g-compute-cli deposit --amount <AMOUNT>
```

Withdrawal

You can withdraw to your wallet with the following command:

```
0g-compute-cli refund --amount <AMOUNT>
```

Note: You can't withdraw the "Lock" amount in the account; only the "Total-Lock" portion can be withdrawn.

View Sub-Account

Sub-accounts are dynamically created when tasks are submitted and used to pay provider service fees. You can view sub-account information with the following command:

```
0g-compute-cli get-sub-account --provider <PROVIDER_ADDRESS>
```

Possible output:

```
  Overview
┌──────────────────────────────────────────────┬──────────────────────────────────────────────┐
│ Field                                        │ Value                                        │
├──────────────────────────────────────────────┼──────────────────────────────────────────────┤
│ Provider                                     │ 0x3C44CdDdB6a900fa2b58                       │
├──────────────────────────────────────────────┼──────────────────────────────────────────────┤
│ Balance (OG)                                 │ 0.000000000179668154                         │
├──────────────────────────────────────────────┼──────────────────────────────────────────────┤
│ Funds Applied for Return to Main Account (OG)  │ 0.000000000179668154                         │
└──────────────────────────────────────────────┴──────────────────────────────────────────────┘
  Details of Each Amount Applied for Return to Main Account
┌──────────────────────────────────────────────┬──────────────────────────────────────────────┐
│ Amount (OG)                                  │ Remaining Locked Time                        │
├──────────────────────────────────────────────┼──────────────────────────────────────────────┤
│ 0.000000000179668154                         │ 23h 58min 34s                                │
└──────────────────────────────────────────────┴──────────────────────────────────────────────┘
  Deliverables
┌──────────────────────────────────────────────────────────────────────────────────────────────┐
│ Root Hash                                                                                    │
├──────────────────────────────────────────────────────────────────────────────────────────────┤
│ 0x24951e897b1203e8aa1692736837f089a95b70390cc02723505e41ebf9                                 │
│ cac70c                                                                                       │
├──────────────────────────────────────────────────────────────────────────────────────────────┤
│ 0x85b3869bcf14569bb41c3d7d499c9a8eb441e6d606bbe3e10e0fac90e5                                 │
│ 7d36a4                                                                                       │
└──────────────────────────────────────────────────────────────────────────────────────────────┘
```

Overview: An overview of the account

- **Provider:**: Address of the provider corresponding to the sub-account

- **Balance:**: Balance of the sub-account. The main account transfers a certain amount to the sub-account based on the task fee every time a task is created.
- **Funds Applied for Return to Main Account:** Amount in the sub-account requested to be returned to the main account

Details of Each Amount Applied for Return to Main Account: Detailed information about amounts requested to be returned to the main account

- **Amount:** Amount requested to be returned to the main account
- **Remaining Locked Time:** Remaining locked time for the return amount to be available in the main account

Deliverables: Deliverables issued by the provider after task completion

- **Root Hash:** Root hash of the model uploaded to storage
- **Access Confirmed:** Indicates whether the user has confirmed download access to the model based on the root hash

Retrieve Funds

The retrieve funds operation returns the balance from sub-accounts to the main account. This operation is asynchronous and will execute after a specific locking period of 24 hours. The lock time ensures provider rights protection, preventing the user from immediately returning the balance to the main account after provider services are rendered and stopping the provider from getting paid.

```
0g-compute-cli retrieve-fund
```

The above command requests the balance from all sub-accounts to be returned to the main account. After the lock-in period elapses, execute the `retrieve-fund` command again to refund all the amounts whose locking period has concluded to the main account. Check the refund status using the View Sub-Account command.

Other Commands

View Task List

You can view the list of tasks submitted to a specific provider using the following command:

```
0g-compute-cli list-tasks  --provider <PROVIDER_ADDRESS>
```

Download Data

You can download previously uploaded datasets using the command below:

```
0g-compute-cli download --data-path <PATH_TO_SAVE_DATASET> --data-root
<DATASET_ROOT_HASH>
```

Cancel a Task

You can cancel a task before it starts running using the following command:

```
0g-compute-cli cancel-task --provider <PROVIDER_ADDRESS> --task <TASK_ID>
```

Note: Tasks that are already in progress or completed cannot be canceled.

Marketplace (coming soon)

Coming soon

0G Storage SDKs

0G offers two Software Development Kits (SDKs) to seamlessly integrate decentralized storage into your applications:

- **Go SDK:** Ideal for backend systems and applications built with the Go programming language.
- **TypeScript SDK:** Perfect for frontend development and JavaScript-based projects.

SDK Features

Both SDKs provide a streamlined interface to interact with the 0G Storage network, enabling you to:

- **Upload and Download Files:** Securely store and retrieve data of various sizes and formats. Note: you can also use the explorers to do so.
- **Manage Data:** List uploaded files, check their status, and control access permissions.
- **Leverage Decentralization:** Benefit from the 0G network's distributed architecture for enhanced data availability, immutability, and censorship resistance.

Quick Start

To get started quickly, check out our starter kits:

- **TypeScript Starter Kit** - Complete examples with Express.js server and CLI tool
- **Go Starter Kit** - Ready-to-use examples with Gin server and CLI tool

Both repositories include working examples, API documentation, and everything you need to start building with 0G Storage.

GO SDK Integration | **TypeScript SDK Integration**

Overview

The 0G Go SDK enables seamless interaction with the 0G decentralized storage network. This guide will walk you through the installation, setup, and usage of the SDK, including examples of key functionalities.

Installation

To install the 0G Storage Client library:

```
go get github.com/0glabs/0g-storage-client
```

First, import the necessary packages:

```
import (
    "context"
    "github.com/0glabs/0g-storage-client/common/blockchain"
    "github.com/0glabs/0g-storage-client/indexer"
    "github.com/0glabs/0g-storage-client/transfer"
)
```

Key Functionalities

Initialization

Create the necessary clients to interact with the network:

```
// Create Web3 client for blockchain interactions
w3client := blockchain.MustNewWeb3(evmRpc, privateKey)
defer w3client.Close()
// Create indexer client for node management
indexerClient, err := indexer.NewClient(indRpc)
if err != nil {
    // Handle error
}
```

Parameters:

- `evmRpc`: Ethereum RPC URL
- `privateKey`: Your Ethereum private key for signing transactions
- `indRpc`: Indexer RPC endpoint

Node Selection

Select storage nodes before performing file operations:

```
```

```
nodes, err := indexerClient.SelectNodes(ctx, segmentNumber,
expectedReplicas, excludedNodes)
if err != nil {
    // Handle error
}
```

Parameters:

- `ctx`: Context for operation management
- `segmentNumber`: Identifies which storage segment to use
- `expectedReplicas`: Number of file copies to maintain (minimum 1)
- `excludedNodes`: List of nodes to exclude from selection

File Upload

Upload files to the network:

```
uploader, err := transfer.NewUploader(ctx, w3client, nodes)
if err != nil {
    // Handle error
}
txHash, err := uploader.UploadFile(ctx, filePath)
if err != nil {
    // Handle error
}
```

Parameters:

- `ctx`: Context for upload operation
- `w3client`: Web3 client instance
- `nodes`: Selected storage nodes
- `filePath`: Path to the file being uploaded

File Hash Calculation

Calculate a file's Merkle root hash before upload, this will be used for identify file from 0G storage:

```
rootHash, err := core.MerkleRoot(filePath)
if err != nil {
    // Handle error
```

```
}
fmt.Printf("File hash: %s\n", rootHash.String())
```

Parameters:

- `filePath`: Path to the file you want to hash

Returns:

- `rootHash`: A unique identifier for the file based on its content
  - Used for file verification
  - Required for downloading files

File Download

Download files from the network:

```
downloader, err := transfer.NewDownloader(nodes)
if err != nil {
    // Handle error
}
err = downloader.Download(ctx, rootHash, outputPath, withProof)
if err != nil {
    // Handle error
}
```

Parameters:

- `ctx`: Context for download operation
- `rootHash`: File's unique identifier (Merkle root hash)
- `outputPath`: Where to save the downloaded file
- `withProof`: Enable/disable Merkle proof verification
  - `true`: Performs verification
  - `false`: Skips verification

Best Practices

1. **Error Handling**: Implement proper error handling and cleanup
2. **Context Management**: Use contexts for operation timeouts and cancellation
3. **Resource Cleanup**: Always close clients when done using `defer client.Close()`
4. **Verification**: Enable proof verification for sensitive files
5. **Monitoring**: Track transaction status for important uploads

Conclusion

The 0G Go SDK provides a robust way to interact with the 0G Storage network, enabling decentralized file storage, data integrity verification, and efficient transaction management. For more detailed information, refer to the official GitHub repository.

Storage CLI

Overview

The 0G Storage CLI acts as your gateway to interact directly with the 0G Storage network. It simplifies the process of uploading and downloading files, as well as managing other aspects of your decentralized storage experience.

If you want more control over the data location and versioning, you can use 0G Storage CLI. This section introduces the 0G Storage CLI in detail, including subcommands for storage and kv operations, in order for users to use through the terminal. You can develop your own scripts, e.g. regular log uploading cron jobs, with the CLI tool.

0G Storage Web Tool

If you want a sample web based tool to upload and download your files and directories, the first simple and straightforward way is to use the Web Tool.

Installation

Download the source code

```
git clone https://github.com/0glabs/0g-storage-client.git
```

Build the Source Code

Command to compile the Go code into an executable binary called `0g-storage-client`, which you will use to run the CLI commands. Make sure you install Go first

```
cd 0g-storage-client
go build
```

Add the binary to GOPATH

```
mv 0g-storage-client ~/go/bin
export PATH=~/go/bin:$PATH
```

Key Commands

Commands and Flags

```
ZeroGStorage client to interact with ZeroGStorage network
Usage:
  0g-storage-client [flags]
  0g-storage-client [command]
Available Commands:
  completion  Generate the autocompletion script for the specified shell
  download    Download file from ZeroGStorage network
  gateway     Start gateway service
  gen         Generate a temp file for test purpose
  help        Help about any command
  indexer     Start indexer service
  kv-read     read kv streams
  kv-write    write to kv streams
  upload      Upload file to ZeroGStorage network
Flags:
      --gas-limit uint       Custom gas limit to send transaction
      --gas-price uint       Custom gas price to send transaction
  -h, --help                 help for 0g-storage-client
      --log-color-disabled   Force to disable colorful logs
      --log-level string     Log level (default "info")
      --web3-log-enabled     Enable log for web3 RPC
Use "0g-storage-client [command] --help" for more information about a
command
```

Help

The command-line help listing is reproduced below for your convenience. The same information can be obtained at any time from your own client by running:

```
0g-storage-client --help
```

File Upload

Uploads a file to the 0G Storage network.

```
./0g-storage-client upload --url <blockchain_rpc_endpoint> --contract
<log_contract_address> --key <private_key> --node
<storage_node_rpc_endpoint> --file <file_path>
```

- **Options:**

- `--url`: The URL of an RPC endpoint to interact with the blockchain where the 0G smart contracts reside.
- `--contract`: The address of the 0G log contract on the blockchain.
- `--key`: Your private key, which is necessary to sign the transaction that initiates the file upload.
- `--node`: The RPC endpoint of a 0G storage node to handle the actual file storage. You can use the team-deployed node at `https://rpc-storage-testnet.0g.ai` or run your own node.
- `--file`: The path to the file you want to upload.

File Download

To download a file from the 0G Storage network.

```
./0g-storage-client download --node <storage_node_rpc_endpoint> --root <file_root_hash> --file <output_file_path>
```

- **Options:**
  - `--node`: The RPC endpoint of a 0G storage node where the file you want to download is stored.
  - `--root`: The root hash of the file, a unique identifier used to locate the file on the network.
  - `--file`: The path where you want to save the downloaded file.

File Download with Verification

Similar to the basic download command, but it additionally requests a proof of data integrity from the storage node, ensuring the downloaded file hasn't been tampered with.

```
./0g-storage-client download --node <storage_node_rpc_endpoint> --root <file_root_hash> --file <output_file_path> --proof
```

Important Considerations

- **Contract Addresses:** You need the accurate contract addresses for the 0G log contract on the specific blockchain you are using. You can find these on the 0G Storage explorer or in the official documentation.
- **File Root Hash:** To download a file, you must have its root hash. This is provided when you upload a file or can be found by looking up your transaction on the 0G Storage explorer.

- **Storage Node RPC Endpoint:** You can use the team-deployed storage node or run your own node for more control and the potential to earn rewards.

Example Usage

```
# Upload a file named "my_document.txt"
./0g-storage-client upload --url https://rpc-testnet.0g.ai --contract
0x123...abc --key 0x456...def --node https://rpc-storage-testnet.0g.ai --
file my_document.txt
# Download a file with root hash "0x789...ghi" and save it as
"downloaded_file.txt"
./0g-storage-client download --node https://rpc-storage-testnet.0g.ai --
root 0x789...ghi --file downloaded_file.txt
```

Indexer

Due to the sharding mechanism, when the entire network data volume increases, an upload or download request for a single file may need to transfer data between multiple different storage nodes, how to find suitable storage nodes may become a problem for ordinary users.

Indexer is a service used to provide storage node queries, it is usually run by groups or individuals who maintain some stable storage nodes. It returns the trusted node list it maintains or the node list discovered through the p2p network to the user.

Remember: The 0G Storage CLI is a tool for interacting with the 0G network. By understanding its commands and options, you can efficiently manage your data stored on this decentralized platform.

0G Data Availability (DA): Integration

To submit data to the 0G DA, you must run a DA Client node and the Encoder node. The DA client interfaces with the Encoder for data encoding and the Retriever for data access.

Maximum blob size

Users can submit data blobs up to 32,505,852 bytes in length, which are then processed, encoded, and distributed across a network of DA nodes. The system employs a sophisticated data processing flow that includes padding, matrix formation, redundant encoding, and signature aggregation.

Fee Market

As the DA user, you pay a fee which is the (BLOB_PRICE) when submitting DA blob data.

Submitting Data

See example here https://github.com/0glabs/0g-da-example-rust/blob/main/src/disperser.proto

Hardware Requirements

The following table outlines the hardware requirements for different types of DA Client nodes:

Node Type	Memory	CPU	Disk	Bandwidth	Additional Notes
DA Client	8 GB	2 cores	-	100 MBps	For Download / Upload
DA Encoder	-	-	-	-	NVIDIA Drivers: 12.04 on the RTX 4090*
DA Retriever	8 GB	2 cores	-	100 MBps	For Download / Upload

Standing up DA Client, Encoder, Retriever

DA Client DA Encoder DA Retriever

DA Client Node Installation

1. Clone the DA Client Node Repo:

```
git clone https://github.com/0glabs/0g-da-client.git
```

2. Build the Docker Image:

```
cd 0g-da-client
docker build -t 0g-da-client -f combined.Dockerfile .
```

3. Set Environment Variables:

Create a file named `envfile.env` with the following content. Be sure you paste in your private key.

```
# envfile.env
COMBINED_SERVER_CHAIN_RPC=https://evmrpc-testnet.0g.ai
COMBINED_SERVER_PRIVATE_KEY=YOUR_PRIVATE_KEY
ENTRANCE_CONTRACT_ADDR=0x857C0A28A8634614BB2C96039Cf4a20AFF709Aa9
COMBINED_SERVER_RECEIPT_POLLING_ROUNDS=180
COMBINED_SERVER_RECEIPT_POLLING_INTERVAL=1s
COMBINED_SERVER_TX_GAS_LIMIT=2000000
COMBINED_SERVER_USE_MEMORY_DB=true
COMBINED_SERVER_KV_DB_PATH=/runtime/
COMBINED_SERVER_TimeToExpire=2592000
DISPERSER_SERVER_GRPC_PORT=51001
BATCHER_DASIGNERS_CONTRACT_ADDRESS=0x0000000000000000000000000000000000001010
BATCHER_FINALIZER_INTERVAL=20s
BATCHER_CONFIRMER_NUM=3
BATCHER_MAX_NUM_RETRIES_PER_BLOB=3
BATCHER_FINALIZED_BLOCK_COUNT=50
BATCHER_BATCH_SIZE_LIMIT=500
BATCHER_ENCODING_INTERVAL=3s
BATCHER_ENCODING_REQUEST_QUEUE_SIZE=1
BATCHER_PULL_INTERVAL=10s
BATCHER_SIGNING_INTERVAL=3s
```

```
BATCHER_SIGNED_PULL_INTERVAL=20s
BATCHER_EXPIRATION_POLL_INTERVAL=3600
BATCHER_ENCODER_ADDRESS=DA_ENCODER_SERVER
BATCHER_ENCODING_TIMEOUT=300s
BATCHER_SIGNING_TIMEOUT=60s
BATCHER_CHAIN_READ_TIMEOUT=12s
BATCHER_CHAIN_WRITE_TIMEOUT=13s
```

4. Run the Docker Node:

```
docker run -d --env-file envfile.env --name 0g-da-client -v
./run:/runtime -p 51001:51001 0g-da-client combined
```

Configuration

Field	Description
`--chain.rpc`	JSON RPC node endpoint for the blockchain network.
`--chain.private-key`	Hex-encoded signer private key.
`--chain.receipt-wait-rounds`	Maximum retries to wait for transaction receipt.
`--chain.receipt-wait-interval`	Interval between retries when waiting for transaction receipt.
`--chain.gas-limit`	Transaction gas limit.
`--combined-server.use-memory-db`	Whether to use mem-db for blob storage.
`--combined-server.storage.kv-db-path`	Path for level db.
`--combined-server.storage.time-to-expire`	Expiration duration for blobs in level db.
`--combined-server.log.level-file`	File log level.
`--combined-server.log.level-std`	Standard output log level.

Field	Description
`--combined-server.log.path`	Log file path.
`--disperser-server.grpc-port`	Server listening port.
`--disperser-server.retriever-address`	GRPC host for retriever.
`--batcher.da-entrance-contract`	Hex-encoded da-entrance contract address.
`--batcher.da-signers-contract`	Hex-encoded da-signers contract address.
`--batcher.finalizer-interval`	Interval for finalizing operations.
`--batcher.finalized-block-count`	Default number of blocks between finalized block and latest block.
`--batcher.confirmer-num`	Number of Confirmer threads.
`--batcher.max-num-retries-for-sign`	Number of retries before signing fails.
`--batcher.batch-size-limit`	Maximum batch size in MiB.
`--batcher.encoding-request-queue-size`	Size of the encoding request queue.
`--batcher.encoding-interval`	Interval between blob encoding requests.
`--batcher.pull-interval`	Interval for pulling from the encoded queue.
`--batcher.signing-interval`	Interval between slice signing requests.
`--batcher.signed-pull-interval`	Interval for pulling from the signed queue.
`--encoder-socket`	GRPC host of the encoder.
`--encoding-timeout`	Total time to wait for a response from encoder.

Field	Description
`--signing-timeout`	Total time to wait for a response from signer.

Run an OP Stack Rollup on 0G DA

Optimism is a lightning-fast Ethereum L2 blockchain, built with the OP Stack.

0G DA is a high-performance data availability layer that can be used with Optimism to provide a cost-effective and secure solution for storing transaction data.

OP 0G implementation

To implement this server specification, 0G DA provides a `da-server` that runs as a sidecar process alongside the OP Stack rollup node. This server connects to a 0G DA client to securely communicate with the 0G DA network.

Below are the requisite steps to deploy an OP Stack rollup on 0G DA, which the following documentation will walk you through:

- Follow the instructions to set up a 0G DA client node
- Set up a 0G DA encoder node
- Deploy a 0G DA Server as shown below
- Deploy the OP Stack components with configuration adjustments as shown below

GitHub Repository

Find the repository for this integration at https://github.com/0glabs/0g-da-op-plasma

The Optimism codebase has been extended to integrate with the 0G DA `da-server`. This server utilizes the 0G DA Open API to efficiently store and retrieve rollup data.

Deployment Steps

1. Deploy DA Server

Run with Docker | **Build from Source**

Build the Docker image:

```
docker build -t 0g-da-op-plasma .
```

Run the Docker container:

Adjust commands and parameters as required for your setup:

```
    docker run -p 3100:3100 0g-da-op-plasma:latest da-server --addr
0.0.0.0 --port 3100 --zg.server rpc_to_a_da_client //default:
127.0.0.1:51001
```

0G DA DA-server accepts the following flags for 0G DA storage using 0G DA Open API

```
    --zg.server    (default: "localhost:51001")
        0G DA client server endpoint

    --addr
        server listening address

    --port
        server listening port
```

2. Deploy DA Client and DA Encoder

For guidance on setting up a 0G DA client and DA Encoder, refer to the DA integration documentation.

3. Deploying OP Stack

Prerequisites

Ensure you have installed the following software.

Software	Version
Git	OS default
Go	1.21.6
Node	^20
just	1.34.0
Make	OS default

Software	Version
jq	OS default
direnv	Latest

Use the following releases while following the guide:

- op-node/v1.9.1
- op-proposer/v1.9.1
- op-batcher/v1.9.1
- op-geth v1.101408.0

Build the Optimism Monorepo

1. Clone and navigate to the Optimism Monorepo:

```
git clone https://github.com/ethereum-optimism/optimism.git
cd optimism
git fetch --tag --all
git checkout v1.9.1
git submodule update --init --recursive
```

2. Check your dependencies

Run the following script and double check that you have all of the required versions installed. If you don't have the correct versions installed, you may run into unexpected errors.

```
./packages/contracts-bedrock/scripts/getting-started/versions.sh
```

3. Compile the necessary packages:

```
make op-node op-batcher op-proposer
make build
```

Build the Optimism Geth Source

1. Clone and navigate to op-geth:

```
```

```
git clone https://github.com/ethereum-optimism/op-geth.git
cd op-geth
git fetch --tag --all
git checkout v1.101408.0
```

2. Compile op-geth:

```
make geth
```

Get Access to a Sepolia Node

For deploying to Sepolia, access an L1 node using a provider like Alchemy (easier) or run your own Sepolia node (harder).

Fill out environment variables

You'll need to fill out a few environment variables before you can start deploying your chain.

1. Enter the Optimism Monorepo

```
cd ~/optimism
```

2. Duplicate the sample environment variable file

```
cp .envrc.example .envrc
```

3. Fill out the environment variable file Open up the environment variable file and fill out the following variables:

```
| Variable Name | Description
|
| ------------- | ------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
--- |
| `L1_RPC_URL`  | URL for your L1 node (a Sepolia node in this case).
|
| `L1_RPC_KIND` | Kind of L1 RPC you're connecting to, used to inform
optimal transactions receipts fetching. Valid options: `alchemy`,
```

```
`quicknode`, `infura`, `parity`, `nethermind`, `debug_geth`, `erigon`,
`basic`, `any`. |
```

Generate addresses

You'll need four addresses and their private keys when setting up the chain:

- The `Admin` address has the ability to upgrade contracts.
- The `Batcher` address publishes Sequencer transaction data to L1.
- The `Proposer` address publishes L2 transaction results (state roots) to L1.
- The `Sequencer` address signs blocks on the p2p network.

You can use `cast wallet` in the `contracts-bedrock` package for key generation:

1. In the Optimism repo, navigate to the contracts-bedrock package:

```
cd ~/optimism/packages/contracts-bedrock
```

2. Generate accounts:

```
./packages/contracts-bedrock/scripts/getting-started/wallets.sh
```

You should see an output similar to:

```
Copy the following into your .envrc file:

# Admin address
export GS_ADMIN_ADDRESS=0x9625B9aF7C42b4Ab7f2C437dbc4ee749d52E19FC
export
GS_ADMIN_PRIVATE_KEY=0xbb93a75f64c57c6f464fd259ea37c2d4694110df57b2e293db82
# Batcher address
export GS_BATCHER_ADDRESS=0xa1AEF4C07AB21E39c37F05466b872094edcf9cB1
export
GS_BATCHER_PRIVATE_KEY=0xe4d9cd91a3e53853b7ea0dad275efdb5173666720b1100866f

# Proposer address
export GS_PROPOSER_ADDRESS=0x40E805e252D0Ee3D587b68736544dEfB419F351b
export
GS_PROPOSER_PRIVATE_KEY=0x2d1f265683ebe37d960c67df03a378f79a7859038c6d634a6

# Sequencer address
export GS_SEQUENCER_ADDRESS=0xC06566E8Ec6cF81B4B26376880dB620d83d50Dfb
```

```
export
GS_SEQUENCER_PRIVATE_KEY=0x2a0290473f3838dbd083a5e17783e3cc33c905539c0121f9
```

Record and securely store these key details. You'll need to fund `Admin`, `Proposer`, and `Batcher` with Sepolia ETH (0.5 ETH for `Admin`, 0.2 ETH for `Proposer`, 0.1 ETH for `Batcher`).

> **NOTE FOR PRODUCTION:**
> Use secure hardware for key management in production environments. cast wallet is not designed for production deployments.

3. Save the addresses

Copy the output from the previous step and paste it into your .envrc file as directed.

Load environment variables

Now that you've filled out the environment variable file, you need to load those variables into your terminal.

1. Enter the Optimism Monorepo

```
cd ~/optimism
```

2. Load the variables with direnv

Next you'll need to allow direnv to read this file and load the variables into your terminal using the following command.

```
direnv allow
```

3. Confirm that the variables were loaded

After running direnv allow you should see output that looks something like the following (the exact output will vary depending on the variables you've set, don't worry if it doesn't look exactly like this):

```
direnv: loading ~/optimism/.envrc
direnv: export +DEPLOYMENT_CONTEXT +ETHERSCAN_API_KEY +GS_ADMIN_ADDRESS
+GS_ADMIN_PRIVATE_KEY +GS_BATCHER_ADDRESS +GS_BATCHER_PRIVATE_KEY
+GS_PROPOSER_ADDRESS +GS_PROPOSER_PRIVATE_KEY +GS_SEQUENCER_ADDRESS
```

```
+GS_SEQUENCER_PRIVATE_KEY +IMPL_SALT +L1_RPC_KIND +L1_RPC_URL
+PRIVATE_KEY +TENDERLY_PROJECT +TENDERLY_USERNAME
```

Core Contract Deployment

Deploy essential L1 contracts for the chain’s functionality:

1. Update `/optimism/packages/contracts-bedrock/deploy-config` and update file `getting_started.json.`

```
cd packages/contracts-bedrock
./scripts/getting-started/config.sh
```

2. Add the following at the bottom of the config generated:

```
  "useAltDA": true,
  "daCommitmentType": "GenericCommitment",
  "daChallengeWindow": 160,
  "daResolveWindow": 160,
  "daBondSize": 1000000,
  "daResolverRefundPercentage": 0
```

Example config (for reference purpose):

```
{
 "l1StartingBlockTag":
"0x0b2b81474a22fc1122bbb3a465985c5cb40dfd8ef18bfe4fc0a9fa47e775d692",
  "l1ChainID": 11155111,
  "l2ChainID": 42069,
  "l2BlockTime": 2,
  "l1BlockTime": 12,
  "maxSequencerDrift": 600,
  "sequencerWindowSize": 3600,
  "channelTimeout": 300,
  "p2pSequencerAddress": "0xd34514056DBE102dF5c24DAf3e78701b502F34c7",
  "batchInboxAddress": "0xff0000000000000000000000000000000042069",
  "batchSenderAddress": "0x4eD53FeB5b06c60368490683e2b317d7C20eC41E",
  "l2OutputOracleSubmissionInterval": 120,
  "l2OutputOracleStartingBlockNumber": 0,
  "l2OutputOracleStartingTimestamp": 1729571268,
  "l2OutputOracleProposer": "0x7d32557e4e79F494836037aD622AaB60125D8323",
  "l2OutputOracleChallenger":
```

```
"0x6A06226C406f7298E1F7CF3F1aa72f3Bc5eF4247",
  "finalizationPeriodSeconds": 12,
  "proxyAdminOwner": "0x6A06226C406f7298E1F7CF3F1aa72f3Bc5eF4247",
  "baseFeeVaultRecipient": "0x6A06226C406f7298E1F7CF3F1aa72f3Bc5eF4247",
  "l1FeeVaultRecipient": "0x6A06226C406f7298E1F7CF3F1aa72f3Bc5eF4247",
  "sequencerFeeVaultRecipient":
"0x6A06226C406f7298E1F7CF3F1aa72f3Bc5eF4247",
  "finalSystemOwner": "0x6A06226C406f7298E1F7CF3F1aa72f3Bc5eF4247",
  "superchainConfigGuardian":
"0x6A06226C406f7298E1F7CF3F1aa72f3Bc5eF4247",
  "baseFeeVaultMinimumWithdrawalAmount": "0x8ac7230489e80000",
  "l1FeeVaultMinimumWithdrawalAmount": "0x8ac7230489e80000",
  "sequencerFeeVaultMinimumWithdrawalAmount": "0x8ac7230489e80000",
  "baseFeeVaultWithdrawalNetwork": 0,
  "l1FeeVaultWithdrawalNetwork": 0,
  "sequencerFeeVaultWithdrawalNetwork": 0,
  "gasPriceOracleOverhead": 0,
  "gasPriceOracleScalar": 1000000,
  "enableGovernance": true,
  "governanceTokenSymbol": "OP",
  "governanceTokenName": "Optimism",
  "governanceTokenOwner": "0x6A06226C406f7298E1F7CF3F1aa72f3Bc5eF4247",
  "l2GenesisBlockGasLimit": "0x1c9c380",
  "l2GenesisBlockBaseFeePerGas": "0x3b9aca00",
  "eip1559Denominator": 50,
  "eip1559DenominatorCanyon": 250,
  "eip1559Elasticity": 6,
  "l2GenesisFjordTimeOffset": "0x0",
  "l2GenesisRegolithTimeOffset": "0x0",
  "l2GenesisEcotoneTimeOffset": "0x0",
  "l2GenesisDeltaTimeOffset": "0x0",
  "l2GenesisCanyonTimeOffset": "0x0",
  "systemConfigStartBlock": 0,
  "requiredProtocolVersion":
"0x0000000000000000000000000000000000000000000000000000000000000000",
  "recommendedProtocolVersion":
"0x0000000000000000000000000000000000000000000000000000000000000000",
  "faultGameAbsolutePrestate":
"0x03c7ae758795765c6664a5d39bf63841c71ff191e9189522bad8ebff5d4eca98",
  "faultGameMaxDepth": 44,
  "faultGameClockExtension": 0,
  "faultGameMaxClockDuration": 1200,
  "faultGameGenesisBlock": 0,
  "faultGameGenesisOutputRoot":
"0x0000000000000000000000000000000000000000000000000000000000000000",
  "faultGameSplitDepth": 14,
```

```
  "faultGameWithdrawalDelay": 600,
  "preimageOracleMinProposalSize": 1800000,
  "preimageOracleChallengePeriod": 300,
  "useAltDA": true,
  "daCommitmentType": "GenericCommitment",
  "daChallengeWindow": 160,
  "daResolveWindow": 160,
  "daBondSize": 1000000,
  "daResolverRefundPercentage": 0
}
```

2. Navigate to `/optimism/packages/contracts-bedrock/` and the deploy contracts (this can take up to 15 minutes):

```
DEPLOYMENT_OUTFILE=deployments/artifact.json \
DEPLOY_CONFIG_PATH=deploy-config/getting-started.json \
forge script scripts/deploy/Deploy.s.sol:Deploy  --broadcast --private-
key $GS_ADMIN_PRIVATE_KEY \
--rpc-url $L1_RPC_URL --slow
```

3. L2 Allocs

```
CONTRACT_ADDRESSES_PATH=deployments/artifact.json
DEPLOY_CONFIG_PATH=deploy-config/getting-started.json
STATE_DUMP_PATH=deploy-config/statedump.json forge script
scripts/L2Genesis.s.sol:L2Genesis --sig 'runWithStateDump()' --chain
<YOUR_L2_CHAINID>
```

Setting Up L2 Configuration

After configuring the L1 layer, focus shifts to establishing the L2 infrastructure. This involves generating three key files:

- genesis.json for the genesis block
- rollup.json for rollup configurations
- jwt.txt for secure communication between op-node and op-geth

1. Navigate to the op-node directory:

```
cd ~/optimism/op-node
```

2. Run the following command, ensuring you replace RPC with your specific L1 RPC URL. This generates the genesis.json and rollup.json files:

```
go run cmd/main.go genesis l2 \
--deploy-config ../packages/contracts-bedrock/deploy-config/getting-started.json \
--l1-deployments ../packages/contracts-bedrock/deployments/artifact.json \
--outfile.l2 genesis.json \
--outfile.rollup rollup.json \
--l1-rpc $L1_RPC_URL \
--l2-allocs ../packages/contracts-bedrock/deploy-config/statedump.json
```

You'll find the newly created genesis.json and rollup.json in the op-node package.

3. Add the following at the end of rollup.json:

```
 "alt_da": {
    "da_challenge_contract_address": "0x0000000000000000000000000000000000000000",
    "da_commitment_type": "GenericCommitment",
    "da_challenge_window": 160,
    "da_resolve_window": 160
  }
```

4. Generate a `jwt.txt` file, which is crucial for the secure interaction between nodes:

```
openssl rand -hex 32 > jwt.txt
```

5. To get op-geth ready, move the genesis.json and jwt.txt files into its directory:

```
cp genesis.json ~/op-geth
cp jwt.txt ~/op-geth
```

Initialize and Configure Geth

Prepare `op-geth` for running the chain:

1. Navigate to op-geth:

```
cd ~/op-geth
```

2. Create a data directory:

```
mkdir datadir
```

3. Initialize with the genesis file:

```
build/bin/geth init --datadir=datadir genesis.json
```

Running op-geth

To initiate `op-geth`, navigate to its directory and execute the following commands:

```
cd ~/op-geth
./build/bin/geth \
  --datadir ./datadir \
  --http \
  --http.corsdomain="*" \
  --http.vhosts="*" \
  --http.addr=0.0.0.0 \
  --http.port=9545 \
  --http.api=web3,debug,eth,txpool,net,engine \
  --ws \
  --ws.addr=0.0.0.0 \
  --ws.port=9546 \
  --ws.origins="*" \
  --ws.api=debug,eth,txpool,net,engine \
  --syncmode=full \
  --nodiscover \
  --maxpeers=0 \
  --networkid=42069 \
  --authrpc.vhosts="*" \
  --authrpc.addr=0.0.0.0 \
  --authrpc.port=9551 \
  --authrpc.jwtsecret=./jwt.txt \
  --rollup.disabletxpoolgossip=true \
  --state.scheme=hash
```

`op-geth` is now active, but block creation will begin once `op-node` is operational.

Running op-node

To launch op-node, which acts as a consensus client, run:

```
cd ~/optimism/op-node
./bin/op-node \
  --l2=http://localhost:9551 \
  --l2.jwt-secret=./jwt.txt \
  --sequencer.enabled \
  --sequencer.l1-confs=5 \
  --verifier.l1-confs=4 \
  --rollup.config=./rollup.json \
  --rpc.addr=0.0.0.0 \
  --rpc.port=8547 \
  --p2p.disable \
  --rpc.enable-admin \
  --p2p.sequencer.key=$GS_SEQUENCER_PRIVATE_KEY \
  --l1=$L1_RPC_URL \
  --l1.rpckind=$L1_RPC_KIND \
  --altda.enabled=true \
  --altda.da-server=<DA_SERVER_HTTP_URL> \
  --altda.da-service=true \
  --l1.beacon.ignore=true
```

Block creation will commence once op-node starts processing L1 information and interfaces with op-geth.

P2P Synchronization

To optimize synchronization and avoid network resource waste:

- Disable p2p sync (--p2p.disable) by default.
- Use specific command line parameters for synchronization among multiple nodes.

Running op-batcher

`op-batcher` is crucial in publishing transactions from the Sequencer to L1. Ensure it has at least 1 Sepolia ETH for operational continuity.

```
cd ~/optimism/op-batcher
./bin/op-batcher \
  --l2-eth-rpc=http://localhost:9545 \
```

```
  --rollup-rpc=http://localhost:8547 \
  --poll-interval=1s \
  --sub-safety-margin=6 \
  --num-confirmations=1 \
  --safe-abort-nonce-too-low-count=3 \
  --resubmission-timeout=30s \
  --rpc.addr=0.0.0.0 \
  --rpc.port=8548 \
  --rpc.enable-admin \
  --max-channel-duration=1 \
  --l1-eth-rpc=$L1_RPC_URL \
  --private-key=$GS_BATCHER_PRIVATE_KEY \
  --altda.enabled=true \
  --altda.da-service=true \
  --altda.da-server=<DA_SERVER_HTTP_URL>
```

Controlling Batcher Costs

The --max-channel-duration=n setting tells the batcher to write all the data to L1 every n L1 blocks. When it is low, transactions are written to L1 frequently and other nodes can synchronize from L1 quickly. When it is high, transactions are written to L1 less frequently and the batcher spends less ETH. If you want to reduce costs, either set this value to 0 to disable it or increase it to a higher value.

Running op-proposer

Finally, start `op-proposer` to propose new state roots:

```
cd ~/optimism/op-proposer
./bin/op-proposer \
  --poll-interval=12s \
  --rpc.port=9560 \
  --rollup-rpc=http://localhost:8547 \
  --l2oo-address=$L2OO_ADDR \
  --private-key=$PROPOSER_KEY \
  --l1-eth-rpc=$L1_RPC
```

Acquire Sepolia ETH for Layer 2

To obtain ETH on your Rollup:

1. Go to `contracts-bedrock`:

```
```

```
cd ~/optimism/packages/contracts-bedrock
```

2. Find the L1 standard bridge contract address:

```
cat deployments/artifact.json | jq -r .L1StandardBridgeProxy
```

3. Send Sepolia ETH to the bridge contract address.

Conduct Test Transactions

You now have a fully operational 0gDA-Powered Optimism-based EVM Rollup. Experiment with it as you would with any other test blockchain.

Congratulations on setting up your chain!

> **NOTE**
>
> - This is a beta integration, and active development is ongoing
> - Ensure all necessary ports are open in your firewall configuration
> - Refer to the Optimism documentation for additional configuration options and troubleshooting

Run an Arbitrum Nitro Rollup on 0G DA

Arbitrum Nitro is a high-performance Ethereum rollup that uses a new consensus mechanism called "Nitro" to achieve scalability and efficiency.

0G DA is a high-performance data availability layer that can be used for Arbitrum Nitro to provide a cost-effective and secure solution for storing transaction data.

To implement this server spec, EigenDA provides EigenDA Proxy which is run as a dependency alongside OP Stack sequencers and full nodes to securely communicate with the EigenDA disperser.

DA provider implementation

The Arbitrum Nitro code includes a DataAvailabilityProvider interface, which is utilized throughout the codebase for storing and retrieving data from various providers, including EIP-4844 blobs, Anytrust, and now 0G.

This integration adds an implementation of the DataAvailabilityProvider interface specifically for Celestia. The main functionality for posting and retrieving data is found in the das/zerogravity.go file, where the data is stored on 0G.

GitHub repository

Find the repository for this integration at https://github.com/0glabs/nitro.

Prerequisites

- 0G DA Client Node
- 0G DA Encoder Node
- 0G Arbitrum Nitro Rollup Kit

> **WARNING:** This is a beta integration and we are working on resolving open issues.

Caldera on 0G DA

Under construction...

0G Faucet

Welcome to the 0G Faucet – your gateway to obtaining testnet tokens for the 0G network!

What is the 0G Faucet?

The 0G Faucet provides free testnet tokens, essential for interacting with the 0G testnet. Whether you're running a node, utilizing our storage or data availability (DA) network, you'll need these tokens to test and explore the 0G environment without using real cryptocurrency.

How to Get Testnet Tokens

- **Use the Faucet Website:** The easiest way to request tokens is through our Official Faucet, Hub Faucet. Each user can receive up to 0.1 OG token per day, which is sufficient for most testing needs.

- **Request via Discord:** If you require more than 0.01 OG token per day, please reach out in our vibrant Discord community to request additional tokens.

> **IMPORTANT**
>
> - Testnet tokens have no real-world value and are for testing purposes only.
> - Ensure you have an EVM-compatible wallet to receive 0G testnet tokens. An example of such is MetaMask
> - There may be a limit to how often you can request tokens.
> - If you encounter any issues, please reach out to our support team on Discord.

Learn More

Curious about how to use your testnet tokens? Check out our run a node section of our documentation for tutorials and examples!

0G Explorers: Visualize the Blockchain Ecosystem

Our explorers provide a user-friendly window into the 0G ecosystem. By offering powerful visualization tools, they eliminate the need for complex CLI commands, making it easier for anyone to understand what's happening on-chain. A well-designed UI enhances your experience, allowing you to focus on insights rather than memorizing specific commands.

These explorers provide detailed insights into storage and chain activities, helping you understand and analyze the network effectively.

Storage Scan

Storage Scan is your go-to tool for exploring storage-related activities within the network. Use Storage Scan to:

- Track data storage and retrieval activities
- Analyze storage capacity and utilization across the network
- View detailed transaction histories and storage metrics

Chain Scan

Chain Scan provides a comprehensive view of 0G chain activity and transactions. Use Chain Scan to:

- View real-time blockchain transactions
- Explore blocks, addresses, and smart contracts
- Analyze network performance and activity trends

Nodes Guru

Nodes Guru provides key information and monitoring tools for validators and node operators to track the health and performance of the network. Use Nodes Guru to:

- View Validator performance metrics like uptime, block proposals, and missed blocks
- Obtain staking and rewards info
- Participate in governance proposals and vote on network upgrades or changes

Security at 0G

At 0G, we prioritize the security and integrity of our platform. Our commitment to security is reflected in our rigorous audit processes and our active bug bounty program.

Audits

We regularly conduct thorough security audits of our smart contracts, protocols, and infrastructure to ensure the highest level of security for our users.

Recent Audits

Date	Auditor	Scope	Report
Aug 2024 - Sept 2024	Halborn	0G Storage	Report
Aug 2024	Zellic	0G Storage and 0G DA	Report

For a complete list of our audits and their detailed reports, please visit our GitHub repository.

0G Labs Bug Bounty Program with Hackenproof

At 0G, we believe in the power of **community-driven security**. Our bug bounty program invites security researchers and developers to help us identify and resolve potential vulnerabilities, ensuring the robustness of our systems.

Scope of the Bug Bounty Program

Our bug bounty program covers:

- Smart Contracts
- Infrastructure
- Protocol

Focus Area

In-Scope Vulnerabilities:

We are interested in vulnerabilities that result in incorrect behavior of the smart contract and could lead to unintended functionality, including:

- Stealing or loss of funds
- Unauthorized transactions
- Transaction manipulation
- Attacks on logic (behavior that deviates from the intended business logic)
- Reentrancy attacks
- Reordering transactions
- Overflows and underflows

Out-of-Scope Vulnerabilities:

The following are out of scope for the bug bounty program:

- Theoretical vulnerabilities without proof or demonstration
- Old compiler versions
- Unlocked compiler version
- Vulnerabilities in imported contracts
- Code style guide violations
- Redundant code
- Gas optimizations
- Best practice issues
- Vulnerabilities exploitable through front-run attacks only

Additionally, the following contracts are out of scope for 0g-storage-contract:

- `cashier`
- `token`
- `reward/OnePoolReward`
- `reward/ChunkDecayReward`
- `uploadMarket`
- `utils/Exponent.sol`

Rewards

Rewards are based on the severity of the discovered vulnerability:

Severity	Reward Range
Critical	$35,000
High	$8000

Severity	Reward Range
Medium	$2000
Low	$500

Program Rules

- Avoid using web application scanners for automatic vulnerability searching which generates massive traffic
- Make every effort not to damage or restrict the availability of products, services, or infrastructure
- Avoid compromising any personal data, interruption, or degradation of any service
- Don’t access or modify other user data, localize all tests to your accounts
- Perform testing only within the scope
- Don’t exploit any DoS/DDoS vulnerabilities, social engineering attacks, or spam
- Don’t spam forms or account creation flows using automated scanners
- In case you find chain vulnerabilities we’ll pay only for vulnerability with the highest severity.
- Don’t break any law and stay in the defined scope
- Any details of found vulnerabilities must not be communicated to anyone who is not a HackenProof Team or an authorized employee of this Company without appropriate permission

Disclosure Guidelines

IMPORTANT

- Do not discuss this program or any vulnerabilities (even resolved ones) outside of the program without express consent from the organization
- No vulnerability disclosure, including partial is allowed for the moment.
- Please do NOT publish/discuss bugs

Eligibility and Coordinated Disclosure

We are happy to thank everyone who submits valid reports which help us improve the security. However, only those that meet the following eligibility requirements may receive a monetary reward:

- You must be the first reporter of a vulnerability.
- The vulnerability must be a qualifying vulnerability

- Any vulnerability found must be reported no later than 24 hours after discovery and exclusively through hackenproof.com
- You must send a clear textual description of the report along with steps to reproduce the issue, include attachments such as screenshots or proof of concept code as necessary.
- You must not be a former or current employee of us or one of its contractors.
- ONLY USE the EMAIL under which you registered your HackenProof account (in case of violation, no bounty can be awarded)
- Provide detailed but to-the point reproduction steps

We look forward to working with the community to enhance 0G's security!

0G Whitepaper

Download PDF

Contribute to 0G Blockchain

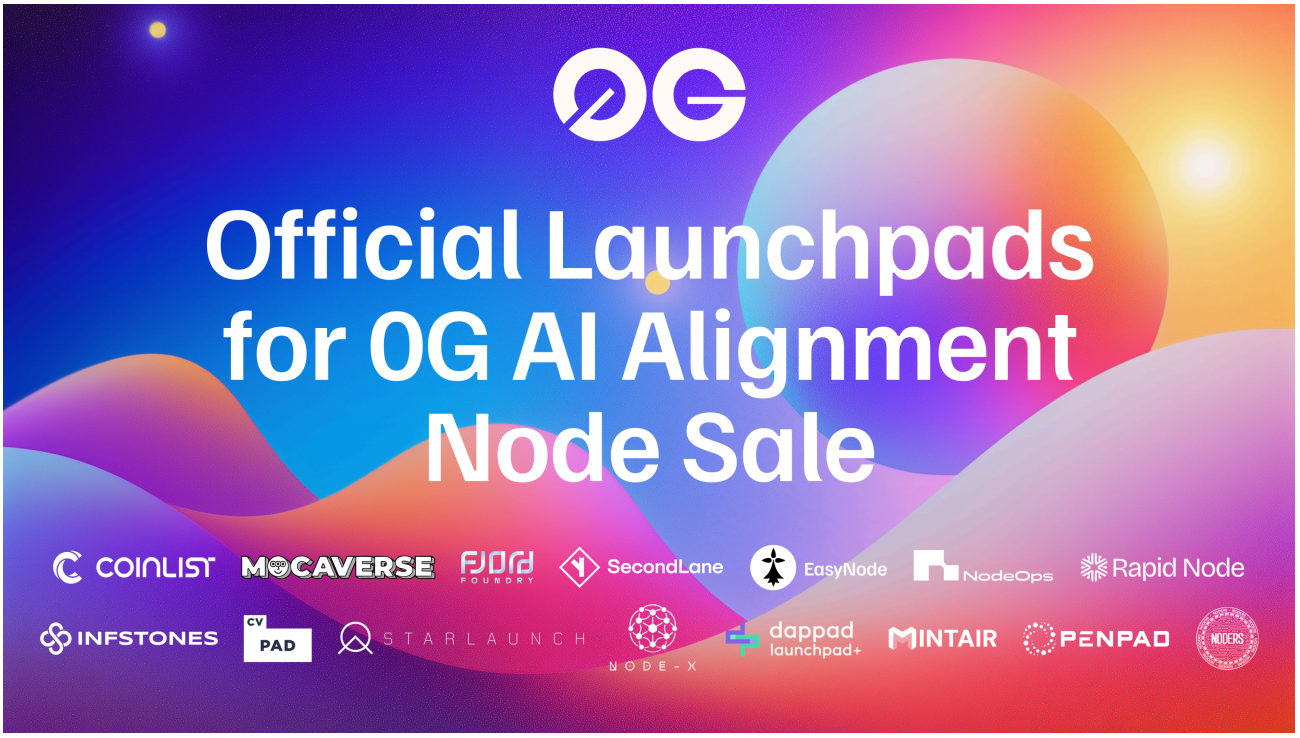
We welcome and encourage you to contribute to our open-source project!

Whether you're a seasoned developer or new to the community, your participation helps us push decentralized AI forward. Every contribution, no matter the size, makes a difference. Please check out our Contribution Guidelines to learn more about our criteria and how you can get involved. All submissions will be reviewed by a team member to ensure they meet our standards.

Let's build the future of decentralized AI together.

Welcome to the 0G Foundation AI Alignment Node Sale

In this section, learn more about the 0G Foundation AI Alignment Node Sale.



Introduction

Purpose and Benefits of the 0G AI Alignment Node Sale

The objective of 0G Foundation: The 0G Foundation’s node sale aims to create a decentralized AI operating system (deAIOS) that operates transparently, safely, and under community influence. Centralized AI structures may lack these attributes, creating potential risks for data integrity and security. The 0G Foundation’s goal is to develop AI as a public good, fostering an ecosystem that prioritizes transparency and reduces reliance on central authorities, making it suitable for broad public utility.

What is an AI Alignment Node?

Alignment nodes provide the key utility of monitoring whether the other kinds of decentralized nodes in the 0G network - validator nodes, storage nodes, security nodes - faithfully follow network protocols. To start, the nodes will have certain utility, and in the near future, AI Alignment Nodes will have additional utility to monitor on-chain AI model drift and to ensure that 0G’s on-chain AI is behaving as intended.

To sustain the node utility operation and allow the network to be further secured, Alignment Node owners may receive a portion of the fees collected by the network by operating the nodes and contributing work to the 0G network. Node sale participants and stakers (i.e., long term believers who secure the ecosystem) may get additional rewards from the 0G ecosystem over time. The 0G node sale will be a launch that enables the community to participate on an equal playing field with entry points within multiple tiers.

Why Run a Node?

Running a node on the 0G network offers participants a chance to directly contribute to the growth and security of our decentralized AI ecosystem. Nodes are the backbone of our network, enabling the following:

- **Decentralization**: Ensure the network remains decentralized and resilient.
- **Network Security**: Enhance the security and reliability of the network by verifying transactions.
- **AI Processing Power**: Support AI computations, thereby contributing to the network's ability to deliver on-chain AI services.
- **Reward Opportunities**: Node operators can earn rewards in the form of native tokens or other incentives for their contribution to the network.

What are the specific use cases for AI Alignment Nodes?

- **Protocol Monitoring**: Ensuring that validator, storage, and security nodes comply with network protocols.
- **AI Model Monitoring**: Tracking AI model alignment and checking for any unintended behavior.
- **Network Security**: Safeguarding the ecosystem by flagging deviations and ensuring consistent ethical standards.
- **Economic and Governance Roles**: Node holders may earn rewards and participate in governance decisions, influencing the network’s direction.

Node Holder Benefits

As an alignment node operator in 0G's ecosystem, users have the opportunity to earn up to 15% of the total 0G ecosystem supply allocated over the next 3 years by helping to secure the network. Nodes are rewarded for assisting in checking and verifying the correct behavior of storage, DA, and serving nodes within the network, playing a crucial role in ensuring data integrity for the AI and other supported workloads. Additional rewards will be provided from 0G’s vibrant and growing ecosystem as a coordinated effort to maintain the AI integrity and security of the platform.

The Alignment Node sale offers holders the chance to join the network and contribute to the development and security of a decentralized ecosystem for the largest collection of decentralized AI computing and data processing power, without requiring every regular user to be prepared for intensive computational tasks from day one.

Sale Structure, Dates, and Tiers

Sale Phases

The sale is structured into two phases:

- **Whitelist Sale**: Whitelist Sale opens November 11, 2024, at 12 PM UTC and remains open for two days. The sale will be denominated in USDC on Arbitrum, and only whitelisted participants may purchase nodes during this phase.
- **Public Sale** opens November 13, 2024, at 12 PM UTC, is denominated in USDC on Arbitrum, and is available to all users, subject to geographic and regulatory restrictions. Please see our disclaimer for more information.

Tier Pricing

The node sale is segmented into 32 pricing tiers, beginning at 0.05 ETH per node, allowing for varied entry points that cater to diverse participant levels. While pegged to an ETH snapshot price of $3,130, both sales are conducted in USDC on Arbitrum. See here for more details.

NODE SALE PRICE

Snapshot ETH Price: November 10, 9 AM UTC @ $3130 Per ETH

Total Supply For Nodes: 15%

Total Nodes Available: 175,500

Tier	Number of Nodes	Node License Price (ETH)	Tier Implied FDV with 50% Participation ($M)	Tier Implied FDV with 100% Participation ($M)
1	3,000	0.050	92	183
2	4,000	0.058	106	212
3	5,000	0.067	123	245
4	6,000	0.078	143	286
5	7,000	0.090	165	330
6	8,000	0.103	189	377
7	8,000	0.118	216	432
8	8,000	0.135	247	494
9	8,000	0.154	282	564
10	7,000	0.175	320	641
11	7,000	0.198	363	725
12	7,000	0.224	410	820
13	7,000	0.254	465	930
14	6,000	0.288	527	1055
15	6,000	0.323	591	1183
16	6,000	0.362	663	1326
17	6,000	0.406	743	1487
18	6,000	0.455	833	1666
19	6,000	0.510	934	1868
20	6,000	0.536	981	1963
21	5,000	0.563	1031	2062
22	5,000	0.592	1084	2168
23	5,000	0.622	1139	2278
24	4,000	0.654	1198	2395
25	4,000	0.687	1258	2516
26	4,000	0.722	1322	2644
27	4,000	0.759	1390	2780
28	3,500	0.797	1459	2919
29	3,500	0.837	1533	3065
30	3,500	0.879	1609	3219
31	3,500	0.923	1690	3380
32	3,500	0.970	1776	3552

Eligibility and Whitelist Participation

- **Whitelist Access**: Community engagement, role-based eligibility, and participation in 0G’s Discord or X channels provide pathways to secure a whitelist spot. Please be aware that joining the whitelist does not guarantee a purchase; it only ensures allocation access during the whitelist phase.
- **Special Roles and Partnerships**: Selected community members, project partners, and Key Opinion Leaders (KOLs) may receive additional access or promotional privileges, subject to guidelines for eligible users and without guarantees for resale or transferability of whitelist spots.

Disclaimer

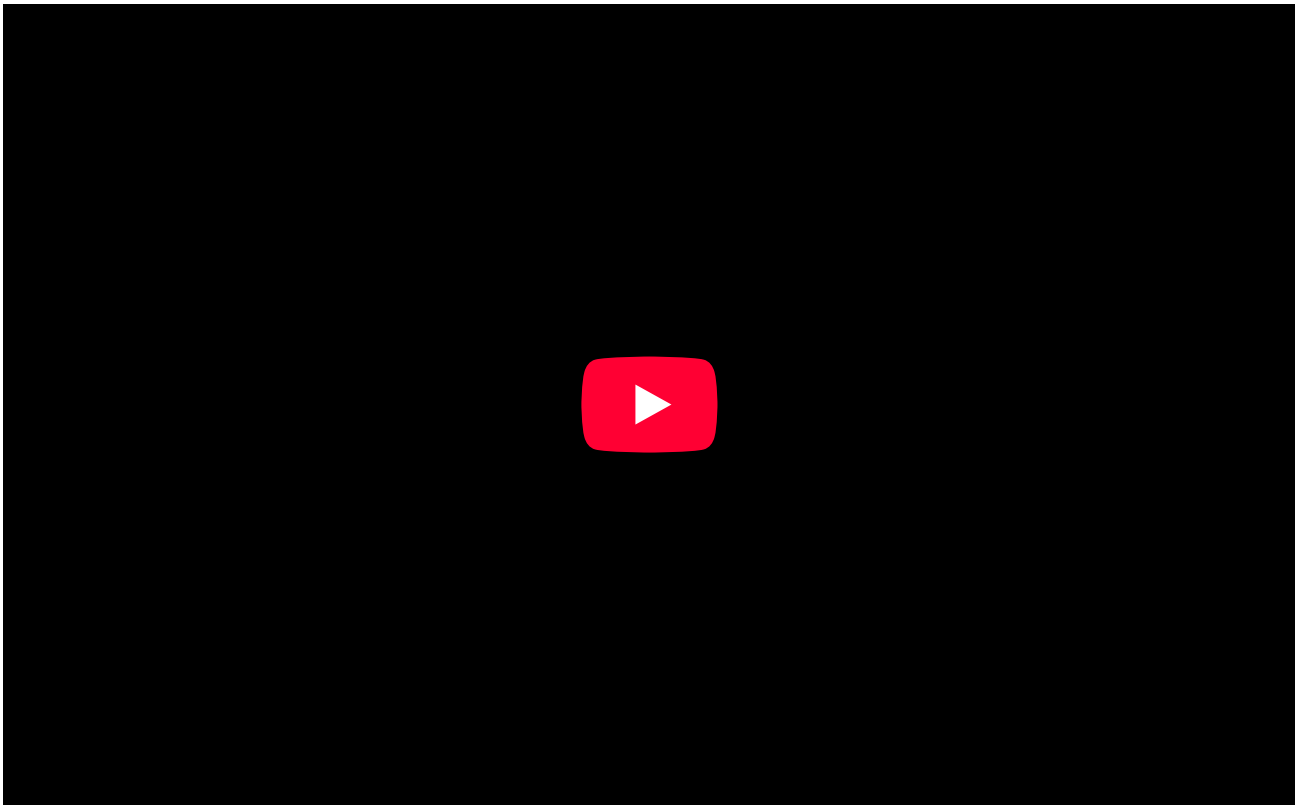
Please review the full details and terms of the 0G Node Sale by visiting our disclaimer.

Purchasing Nodes: Steps and Payment Options

Supported Blockchains and Payment Methods: The sale will be conducted in USDC on Arbitrum, but we provide a live bridging gateway at checkout that supports multiple blockchains (ETH, Arbitrum, BNB) and tokens. All purchases require a compatible wallet, and the resulting node licenses are issued as non-transferable ERC-721 NFTs.

Video Tutorial

> **IMPORTANT**
> Please note: Initially, the Public Sale was intended to use wETH on Arbitrum, but after the community's feedback, it will be USDC on Arbitrum. Please note this correction to USDC on Arbitrum as the video says wETH.



Note that this video is for demonstrative purposes only, may not reflect exact details and have different prices or tokens than the actual sale.

Step-by-Step Purchase Process:

Step 1 - Go to https://node.0gfoundation.ai and connect your wallet

Step 2 - Select the chain and token you want to purchase

Step 3 - View the tiers available and purchase

In total there are 32 available for purchase. If you hold a whitelisted address, the Tier that you are entitled to will be shown on the top. Sold out tiers will be moved to the bottom of the page.

Step 4 - Set the quantity

Start by inputting the number of node(s) that you will be purchasing.

Step 5 - Input Promo Code (Optional)

If you have a Promo Code, input before your purchase. The discount rate will be applied directly to your checkout price. Additionally, if you have already purchased a node, you can provide your wallet address as a referral promo code too! This will provide a 10% rebate to the referral, and 10% commission if they purchase successfully.

Step 6 - Approve transaction

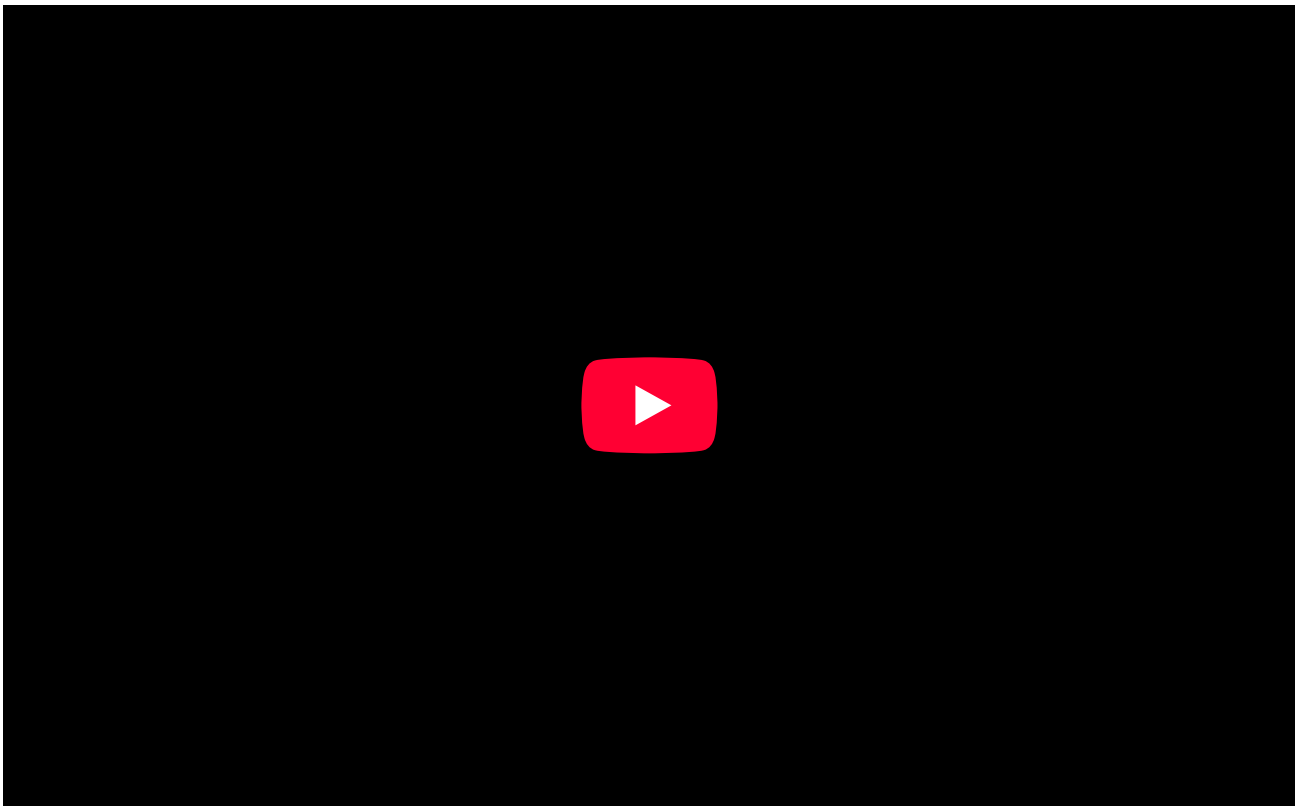
Once you are confirmed on the final price, click the "Approve" button. You are then required to approve the transaction via your wallet.

Step 7 - Complete purchase

Continue by clicking "Purchase". You will need to confirm the transaction via your wallet. After confirming your purchase, you will receive a notification on the top right of the page to inform you of the successful purchase.

Incentives, Rewards, and Vesting Mechanisms

Rebate/Commission Portal Tutorial



How can node buyers create a referral code?

Node buyers will be able to share their wallet address as the referral code after they made a purchase, it would give a 10% rebate to their referrals.

What rebate is available when using a referral code?

If you enter a referral code when purchasing a node, you'll receive a 10% rebate on the total price.

How will I receive my commission? (For Referrers)

You will be able to claim the commission of any successful node purchased through the node on the reward claim site, which will be available after the public sale. Please refer to the 0G X for access to the claim site closer to the sale date.

Vesting Terms

Rewards from node operation vest over a three-year schedule, promoting consistent and long-term engagement. Vesting reduces the likelihood of short-term sales, fostering network stability and growth.

Additional Community Incentives

Partnerships, 0G ecosystem communities, and referrals provide extra benefits, such as rebates or promotional whitelist access. All promotional activities adhere to regulatory guidelines, and any reward or referral payments are conditional upon KYC eligibility.

Compliance and Regulatory Requirements

Regulation S Compliance

The sale follows Regulation S guidelines, restricting U.S. persons from participating. The sale website, promotional content, and user interface clearly indicate these restrictions, and KYC verification is mandatory for claiming rewards to maintain regulatory adherence. Prospective participants are advised to review these conditions and understand that resale of node licenses is not permitted within the first 12 months.

Information Disclosure

All communications related to the sale are made with transparency but exclude any directed selling to U.S. persons. Information shared in marketing materials, promotional activities, and social media avoids U.S.-targeted content, aligning with compliance requirements to mitigate any regulatory risks.

Frequently Asked Questions

Where do I purchase AI Alignment Nodes?

https://node.0gfoundation.ai/

How do I purchase AI Alignment Nodes?

See the step by step guide here.

Who can participate in the Node Sale?

The 0G AI Alignment Node Sale is open to community members in eligible regions who meet the necessary criteria. Please review all disclaimers and important details regarding the 0G Node Sale by visiting our disclaimer here. Only persons who meet the requirements for the sale are allowed to participate. KYC will be required from the purchaser before any receipt of rewards.

Are Whitelist and Public Allocations Separate?

The whitelist allocation is independent and will not be rolled over to the public sale. The **Whitelist Sale** opens November 11, 2024, at 12 PM UTC. The sale will be denominated in USDC on Arbitrum, and only whitelisted participants may purchase nodes during this phase. The **Public Sale** opens November 13, 2024, at 12 PM UTC, is denominated in USDC on Arbitrum, and is available to all users, subject to geographic and regulatory restrictions. Please see our disclaimer for more information.

What price will the sale be pegged to?

Both the Whitelist and Public Sales are denominated in USDC on Arbitrum. Node pricing will be pegged to a snapshot price of wETH of $3,130.

When Can I Operate My Node?

Node operations will commence after the mainnet launch, projected for Q1 2025. License holders will receive further instructions for operation and delegation options if preferred.

How Do Rewards for Alignment Nodes Compare with Other Nodes?

Alignment Nodes are expected to offer higher reward rates due to their unique responsibilities, limited supply, and operational requirements compared to storage or validator nodes.

Are Multiple NFTs Supported per Server?

Yes, users may operate multiple NFTs on a single server, with allowances for connecting and managing several nodes under one setup.

Whitelist & Node Sale

What is the whitelist (WL)?

Whitelist is a pre-approved list of participants who are given exclusive access to certain privileges during a sale event. This system is used to reward and incentivize key contributors, partners, or early supporters of a project.

Does entering the whitelist guarantee that I can definitely purchase a node?

Your whitelist guarantees an allocation during the Whitelist sale period (starting Nov 11). If a purchase is not made within this period, your allocation will be released.

How to join the whitelist?

To get a whitelist spot for the 0G Foundation Node Sale, community members and ecosystem participants are eligible for allocations, please visit 0G X and Discord for ways to receive a whitelist. You can also apply for a whitelist spot by filling out the whitelist form here.

Payment & Licenses

What payment methods will be accepted for purchasing a node?

The nodes will be priced in USDC for both the Whitelist Sale and Public Sale, both on Arbitrum. However, to facilitate payment for users on different chains, the AI Alignment Node Sale will be accepting multiple tokens across multiple networks through a live bridging aggregator that accepts multichain payment including but not limited to BTC, ETH, ARB, SOL, etc. More info here.

How will the node licenses be distributed?

After the node sale period is completed, node licenses will be distributed as an NFT to the purchase wallet of the user.

What will I receive from participating in the node sale?

You will receive a soulbound NFT (ERC-721) which represents your Node License. The NFTs can be minted and transferred to your wallet via claim.0gfoundation.ai after the conclusion of the node sale. You will be able to operate the node after 0G Mainnet is live.

Will the NFTs be transferable?

The Alignment Node license gives buyers lifetime access. The NFTs will be non-transferable for the first year after the node sale.

Node Operations

What are the hardware requirements?

0G Foundation’s AI Alignment nodes are designed for adoption - they can be run on community member's laptops, desktops, mobiles, or even on cloud instances. As for device requirements, the configuration needed is very minimal:

- 64MB RAM
- 1 x86 CPU Core @2.1GHz
- 10GB Disk Space
- 10Mbps Internet Connection

When can I begin operating my node?

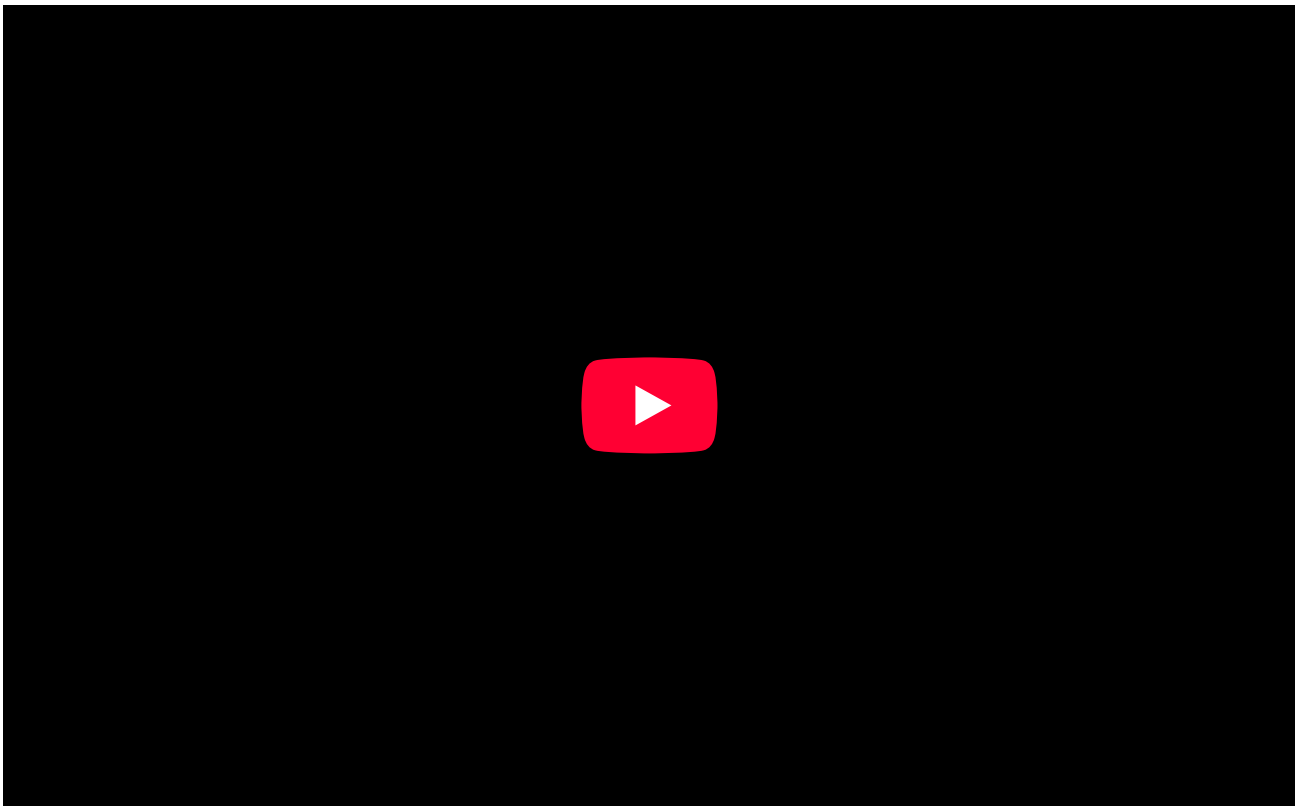
AI Alignment utility will go live in 2025, after 0G Mainnet Launch.

How many nodes can I purchase?

The number of purchasable nodes will be capped per tier. Please refer to the Node Sale Tier documentation for reference.

How do I run a node? Is it complicated?

Running a node can be quite straightforward and easy, typically involving just a few steps. Here's a video tutorial to guide you through the process:



If you prefer not to manage the node yourself, you can delegate to other node operators with just a single click through our explorer, which will be available shortly.

Will the sale be accessible from other platforms?

Both Public and Whitelist sale will be available here except for Partnered Launchpads, in which case the front end will be on 0G Partners' website.

What will happen to any unsold node rewards?

Rewards from unsold nodes will be reallocated to the sold node runners.

What is your tokenomics?

What is the % unlocked at TGE?

33% of node rewards will be initially claimable on TGE. In order to encourage long term participation in the 0G ecosystem, there will be a penalty based on duration in which a participant chooses to receive this initial reward. The remaining 67% of alignment rewards are linearly unlocked (daily) over 36 months. This penalty mechanism is subject to community vote, further showcasing the "community first approach" of 0G.

When is KYC required? How is it conducted?

KYC is required before claiming rewards. It will be provided by a third-party provider. No refund will be offered if the node purchaser does not meet KYC requirements.

How many nodes can I purchase?

Each individual can purchase up to a certain amount of nodes per tier. See here for the caps per tier.

What happened to unsold nodes?

Unsold alignment node NFT licenses will be burned. The rewards from the unsold nodes will be re-allocated to the sold verified node buyers.

What are the launchpad partners working on the 0G Foundation Node Sale?

Node Disclaimer

PLEASE READ THE ENTIRETY OF THIS "LEGAL DISCLAIMER" SECTION CAREFULLY. NOTHING HEREIN CONSTITUTES LEGAL, FINANCIAL, BUSINESS, OR TAX ADVICE AND YOU ARE STRONGLY ADVISED TO CONSULT YOUR OWN LEGAL, FINANCIAL, TAX, OR OTHER PROFESSIONAL ADVISOR(S) BEFORE ENGAGING IN ANY ACTIVITY IN CONNECTION HEREWITH. NEITHER 0G FOUNDATION ("THE COMPANY"), ANY OF THE PROJECT CONTRIBUTORS WHO HAVE WORKED ON THE ZEROGRAVITY NETWORK (AS DEFINED HEREIN) OR ANY PROJECT TO DEVELOP THE ZEROGRAVITY NETWORK IN ANY WAY WHATSOEVER, ANY DISTRIBUTOR AND/OR VENDOR OF NODE LICENSE (OR SUCH OTHER RE-NAMED OR SUCCESSOR TICKER CODE OR NAME OF SUCH TOKENS) (THE "DISTRIBUTOR"), NOR ANY RELATED SERVICE PROVIDER SHALL BE LIABLE FOR ANY KIND OF DIRECT OR INDIRECT DAMAGE OR LOSS WHATSOEVER WHICH YOU MAY SUFFER IN CONNECTION WITH ACCESSING ANY MATERIAL RELATING TO NODE LICENSE NFT (THE TOKEN DOCUMENTATION) AVAILABLE ON THE WEBSITE AT https://0g.ai/node-sale (THE WEBSITE, INCLUDING ANY SUB-DOMAINS THEREON) OR ANY OTHER WEBSITES OR MATERIALS PUBLISHED OR COMMUNICATED BY THE COMPANY OR ITS REPRESENTATIVES FROM TIME TO TIME.

1. **Project purpose**: You agree that you are acquiring Node License NFT to participate in the ZeroGravity Network and to obtain services on the ecosystem thereon. The Company, the Distributor, and their respective affiliates would develop and contribute to the underlying source code for the ZeroGravity Network. The Company is acting solely as an arms' length third party in relation to the Node License NFT distribution and is not acting in the capacity as a financial advisor or fiduciary of any person with regard to the distribution of Node License NFT.
2. **Eligibility**: To be eligible to use the Website (including services thereon) or participate in the Node License NFT distribution you must be at least eighteen (18) years of age or older. The Website, interface and services thereon is strictly NOT offered to persons or entities who reside in, are citizens of, are incorporated in, or have a registered office in any Restricted Territory, as defined below (any such person or entity from a Restricted Territory shall be a Restricted Person). If you are a Restricted Person, then do not attempt to access or use the Website. Use of a virtual private network (e.g., a VPN) or other means by Restricted Persons to access or use the Website, interface or services is prohibited. For the purpose of these Terms, Restricted Territory shall mean Belarus, Canada, Cuba, North Korea, Iran, Russia, Syria, the United States, the United Kingdom, Venezuela, Yemen, and specific regions in Ukraine and Russia, namely the Crimea region, Donetsk People's Republic (DNR), Luhansk People's Republic (LNR), as well as the Kherson and Zaporizhia regions.

3. **Validity of Token Documentation and Website**: Nothing in the Token Documentation or the Website constitutes any offer by the Company, the Distributor, or the ZeroGravity Network team to sell any Node License NFT (as defined herein) nor shall it or any part of it nor the fact of its presentation form the basis of, or be relied upon in connection with, any contract or purchase decision. Nothing contained in the Token Documentation or the Website is or may be relied upon as a promise, representation or undertaking as to the future performance of the ZeroGravity Network. The agreement between the Distributor (or any third party) and you, in relation to any distribution or transfer of Node License NFT, is to be governed only by the separate terms and conditions of such agreement.
4. **Deemed Representations and Warranties**: By accessing the Token Documentation or the Website (or any part thereof), you shall be deemed to represent and warrant to the Company, the Distributor, their respective affiliates, and the ZeroGravity Network team as follows:
    - in any decision to acquire any Node License NFT, you have not relied and shall not rely on any statement set out in the Token Documentation or the Website.
    - you shall at your own expense ensure compliance with all laws, regulatory requirements, and restrictions applicable to you (as the case may be).
    - you acknowledge, understand and agree that Node License NFT may have no value, there is no guarantee or representation of value or liquidity for Node License NFT, and Node License NFT is not an investment product nor is it intended for any investment purposes, speculative or otherwise.
    - none of the Company, the Distributor, their respective affiliates, and/or the ZeroGravity Network team shall be responsible for or liable for the value of Node License NFT, the transferability and/or liquidity of Node License NFT, and/or the availability of any market for Node License NFT through third parties or otherwise.
5. The Company, the Distributor, and the ZeroGravity Network team do not and do not purport to make, and hereby disclaims, all representations, warranties or undertaking to any entity or person (including without limitation warranties as to the accuracy, completeness, timeliness, or reliability of the contents of the Token Documentation or the Website, or any other materials published by the Company or the Distributor). To the maximum extent permitted by law, the Company, the Distributor, their respective affiliates, and service providers shall not be liable for any indirect, special, incidental, consequential, or other losses of any kind, in tort, contract, or otherwise (including, without limitation, any liability arising from default or negligence on the part of any of them, or any loss of revenue, income or profits, and loss of use or data) arising from the use of the Token Documentation or the Website, or any other materials published, or its contents (including without limitation any errors or omissions) or otherwise arising in connection with the same. Prospective acquirors of Node License NFT should

carefully consider and evaluate all risks and uncertainties (including financial and legal risks and uncertainties) associated with the distribution of Node License NFT, the Company, the Distributor, and the ZeroGravity Network team.

6. **Node License NFT**: Node License NFT are designed to be utilized, and that is the goal of the Node License NFT distribution. In particular, it is highlighted that Node License NFT:
    - does not have any tangible or physical manifestation and does not have any intrinsic value/pricing (nor does any person make any representation or give any commitment as to its value).
    - is non-refundable, not redeemable for any assets of any entity or organization, and cannot be exchanged for cash (or its equivalent value in any other digital asset) or any payment obligation by the Company, the Distributor, or any of their respective affiliates.
    - does not represent or confer on the token holder any right of any form with respect to the Company, the Distributor (or any of their respective affiliates), or their revenues or assets, including without limitation any right to receive future dividends, revenue, shares, ownership right or stake, share or security, any voting, distribution, redemption, liquidation, proprietary (including all forms of intellectual property or license rights), right to receive accounts, financial statements or other financial data, the right to requisition or participate in shareholder meetings, the right to nominate a director, or other financial or legal rights, equivalent rights, or intellectual property rights, or any other form of participation in or relating to the ZeroGravity Network, the Company, the Distributor, and/or their service providers.
    - is not intended to represent any rights under a contract for differences or under any other contract the purpose or intended purpose of which is to secure a profit or avoid a loss.
    - is not intended to be a representation of money (including electronic money), payment instrument, security, commodity, bond, debt instrument, unit in a collective investment, managed investment scheme, or any other kind of financial instrument or investment.
    - is not a loan to the Company, the Distributor or any of their respective affiliates, is not intended to represent a debt owed by the Company, the Distributor, or any of their respective affiliates, and there is no expectation of profit nor interest payment.
    - does not provide the token holder with any ownership or other interest in the Company, the Distributor, or any of their respective affiliates.
7. Notwithstanding the Node License NFT distribution, users have no economic or legal right over or beneficial interest in the assets of the Company, the Distributor, or any of their affiliates after the token distribution. To the extent any secondary market or exchange for trading Node License NFT does develop, it would be run and operated wholly independently of the Company, the Distributor, and the distribution of Node License NFT. Neither the Company nor

the Distributor will create such secondary markets nor will either entity act as an exchange for Node License NFT.

8. **English language**: The Token Documentation and the Website may be translated into a language other than English for reference purpose only and in the event of conflict or ambiguity between the English language version and translated versions of the Token Documentation or the Website, the English language versions shall prevail. You acknowledge that you have read and understood the English language version of the Token Documentation and the Website.
9. **No Distribution**: No part of the Token Documentation or the Website is to be copied, reproduced, distributed, or disseminated in any way without the prior written consent of the Company or the Distributor. By attending any presentation on this Token Documentation or by accepting any hard or soft copy of the Token Documentation, you agree to be bound by the foregoing limitations.