

Gaia-X Hackathon #4

Gaia-X DLT Network: Including DLT components in the Gaia-X Framework of Self-Descriptions and Federation Services

Albert Peci, Frederic Schwill, Alexander Eger, Kai Meinke, Moritz Kirstein – deltaDAO AG

Monday, 20 June 2022

01

Gaia-X Testnet Overview

What is the Gaia-X Testnet?



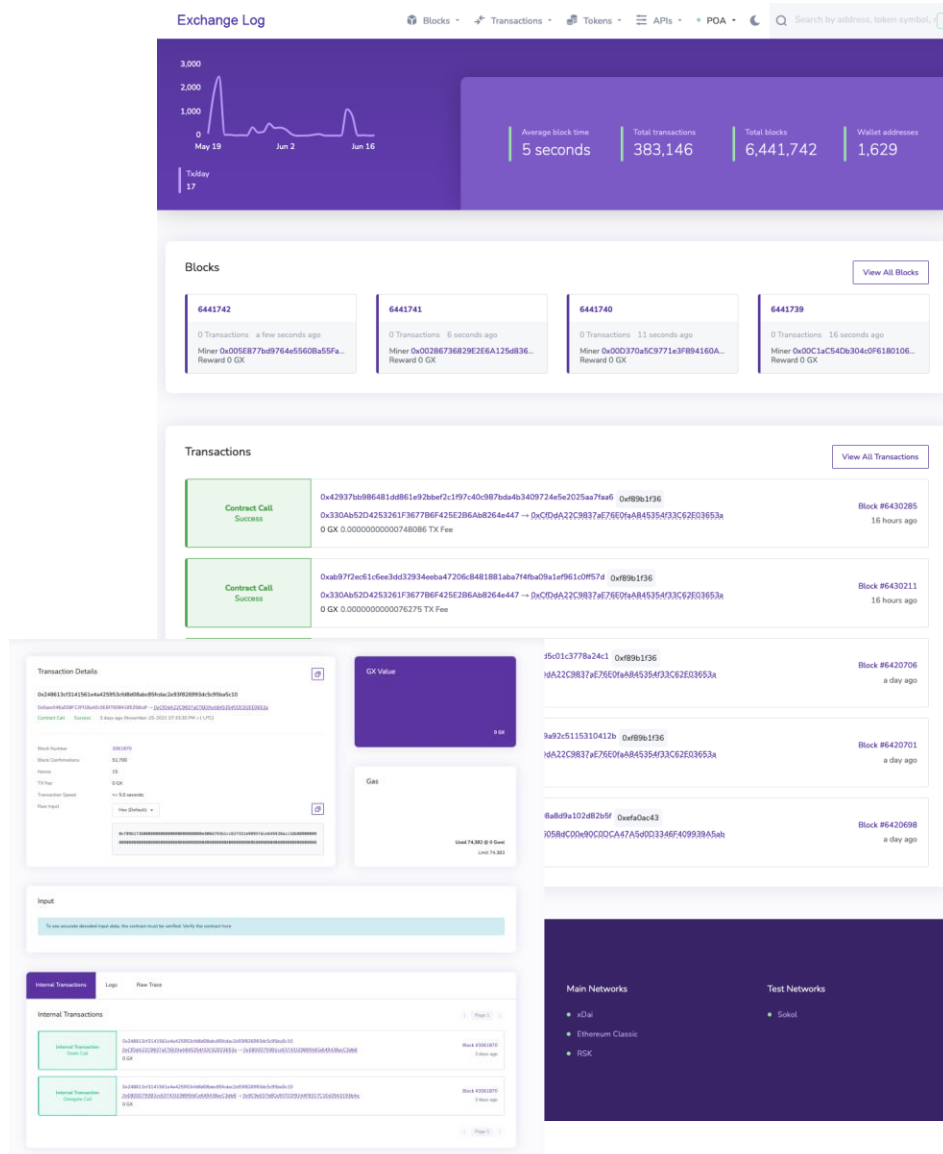
- Gaia-X Testnet is an Open Ethereum based **Proof of Authority** network
- **Aura** (*Authority Round*) is used as consensus algorithm
- It is **Ethereum Virtual Machine (EVM)** compatible
- It is a public-permissioned distributed ledger. All transactions are transparent and can be viewed at <https://exchangelog.minimal-gaia-x.eu/>, but only selected nodes are responsible for consensus.
- Due to its open nature, everyone can run their own archive node and analyze the network. See our stats example at <https://stats.minimal-gaia-x.eu/>
- Everyone can submit transactions. Fees are paid with a custom Gaia-X Test Token. They can be retrieved via a Faucet: <https://faucet.gx.gaiaxtestnet.oceanprotocol.com/>

What does Proof of Authority (PoA) mean?



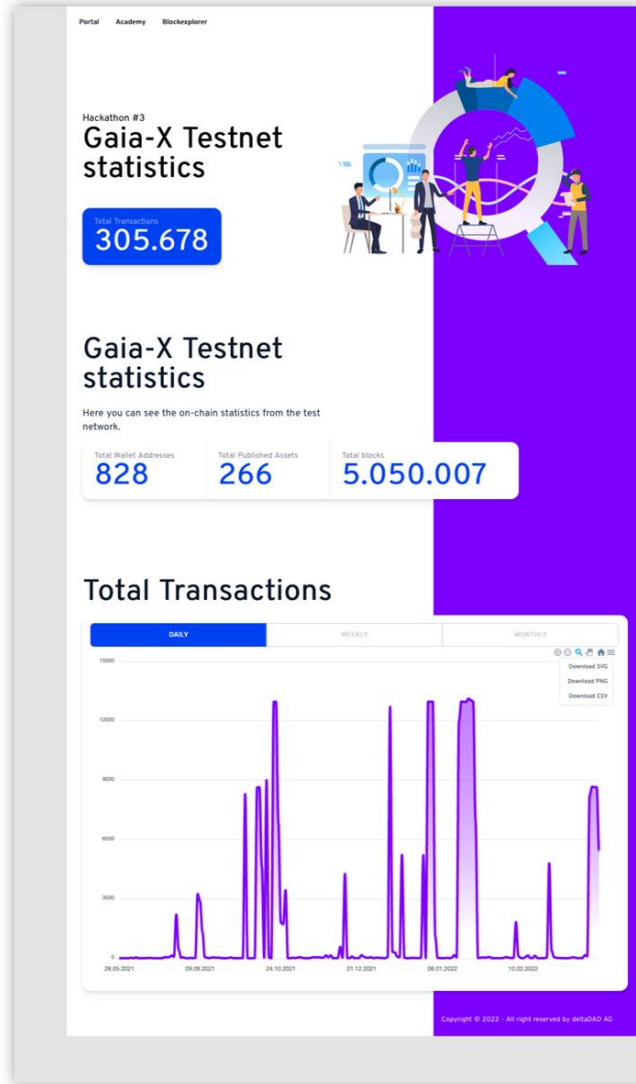
- PoA is a replacement for Proof-of-Work (PoW), which can be used for private chain setups. It is a crucial element to reach consensus in a distributed ledger.
- It does not depend on nodes solving arbitrarily difficult mathematical problems (like Proof-of-Work), but instead uses a set of “authorities”.
- For consortium setting there are no disadvantages of PoA network as compared to PoW: less computationally intensive, more performant due to lower transaction acceptance latency and more predictable (blocks are issued at steady time intervals)
- **Aura (*Authority Round*)** is one of the Blockchain consensus algorithms available in OpenEthereum.
- On each step, each honest node will propagate the chain with the highest score it knows about to all other nodes.

Gaia-X Exchange Logging Service Explorer



- Gaia-X test network Explorer offers an interface to investigate and analyze the Gaia-X test network transactions
- Explorers usually use archive nodes to sync with the network
- Usually, a custom database is used to optimize for queries like address, token symbols, names, transactions hashes and block numbers
- The current explorer distinguishes between Contract Calls, Token Transfers, and failed transactions.
- Explorer: <https://exchangelog.minimal-gaia-x.eu/>

Gaia-X Testnet Stats



Gaia-X Testnet Stats uses a custom script to analyze the distributed ledger for transactions and published assets.

Total Transactions

382.808

Total Wallet Addresses

1.418

Total Published Assets

637

Total blocks

6.441.627

Repositories

<https://stats.minimal-gaia-x.eu/>

<https://github.com/deltaDAO/gaia-x-testnet-statistics-api>

<https://github.com/deltaDAO/gaia-x-testnet-statistics>

<https://github.com/deltaDAO/gaia-x-snapshot>

02

How to Setup an Archive
Node and Describe it?

Gaia-X Testnet Client



- **OpenEthereum** is an implementation of the Ethereum protocol written in Rust, a systems programming language. It is developed and actively maintained by OpenEthereum DAO.
- OpenEthereum is used as a **client** to run **EVM compatible distributed ledgers**.
- Gaia-X Testnet OpenEthereum **client requirements**:
 - multi-core CPU, 8GB RAM and an SSD drive and at least 200GB free space and a decent DSL connection is required
- A list of **boot nodes**, a **config.toml** to connect to the network and a **chain.spec** to define the properties of the chain that the client should connect to:
<https://github.com/oceanByte/gaia-x-node-setup>

Gaia-X Testnet Client Setup



Create a new folder

> **mkdir gaia-x-node && cd gaia-x-node**

Clone config files

> **git clone <https://github.com/oceanByte/gaia-x-node-setup>**

Download the openethereum client <https://github.com/openethereum/openethereum/releases>

> **wget <https://github.com/openethereum/openethereum/releases/download/v3.2.6/openethereum-linux-v3.2.6.zip>**

unzip the client

> **unzip openethereum-linux-v3.2.6.zip**

Run your client with the config file

> **chmod +x openethereum**

> **sudo ./openethereum --config config.toml --reserved-peers bootnodes.txt**

Gaia-X Testnet Config



```
1  [parity]
2  chain = "./chain.spec"
3
4  [network]
5  warp = false
6
7  [rpc]
8  apis = ["web3","eth","net","parity","traces"]
9  processing_threads = 8
10 server_threads = 16
11 interface = "0.0.0.0"
12 cors=["all"]
13
14 [websockets]
15 port = 8546
16 interface = "0.0.0.0"
17 max_connections = 1000
18 apis = ["web3","eth","net","parity","pubsub","traces"]
19 origins = ["all"]
20 hosts = ["all"]
21
22 [footprint]
23 tracing = "on"
24 pruning = "archive"
25 fat_db = "on"
26 cache_size_db = 12000
```

- The OpenEthereum client can be configured via a config.toml
- You can define exactly who and how someone can query data from your node
- Nodes can be used in very different scenarios ranging from archive nodes for data analyses and applications like Explorers to light nodes used to transmit transactions into the network to validator nodes responsible for consensus
- Find a full list of config settings here:
<https://openethereum.github.io/Configuring-OpenEthereum>

Gaia-X Testnet Sync



```
17:38:08 main & chwd +x openethereum
17:38:13 main & sudo ./openethereum --config config.toml --reserved-peers bootnodes.txt
Password:
Loading config file from config.toml
2022-06-19 17:38:23 Starting OpenEthereum/v3.3.5-stable-6c2d892d8-20220405/x86_64-macos/rustc1.58.1
2022-06-19 17:38:23 Keys path /Users/morpheus/Library/Application Support/OpenEthereum/keys/Gaia-X
2022-06-19 17:38:23 DB path /Users/morpheus/Library/Application Support/OpenEthereum/chains/Gaia-X/db/67f641229d32dabc
2022-06-19 17:38:23 State DB configuration: archive +Fat +Trace
2022-06-19 17:38:23 Operating mode: active
2022-06-19 17:38:23 Not preparing block; cannot sign.
2022-06-19 17:38:23 Configured for Gaia-X using AuthorityRound engine
2022-06-19 17:38:23 Listening for new connections on 0.0.0.0:8546.
2022-06-19 17:38:25 Not preparing block; cannot sign.
2022-06-19 17:38:28 Public node URL: enode://b754ec9f3f9372ebdcf1720ad6642fe0b5b3a1ee21b848b3575640db2e319151d794447b0772d2846c641999b2534738f9ac13d0020049f889ad743ff20192.168.178.170:30303
2022-06-19 17:38:28 Syncing #138345 0x71a40f3c219215255717fee06eeeb52ac1d6516ff89f6281d3eb47c6318ed215 1234.72 blk/s 0.0 tx/s 0.0 Mgas/s 0+ 0 Qtd LI:#138345 1/25 peers 2 M8B chain 0 bytes queue RPC: 0 conn, 0 req/s, 0 us
2022-06-19 17:38:33 Syncing #146346 0x4d98b0791 1597.01 blk/s 0.0 tx/s 0.0 Mgas/s 0+ 0 Qtd LI:#146346 1/25 peers 6 M8B chain 0 bytes queue RPC: 0 conn, 0 req/s, 0 us
2022-06-19 17:38:38 Syncing #154983 0x55a94ae2 1725.09 blk/s 0.2 tx/s 0.3 Mgas/s 0+ 32 Qtd LI:#155109 1/25 peers 6 M8B chain 48 KIB queue RPC: 0 conn, 0 req/s, 0 us
2022-06-19 17:38:43 Syncing #164126 0xade4e2db9 1824.52 blk/s 0.0 tx/s 0.0 Mgas/s 0+ 0 Qtd LI:#164253 1/25 peers 5 M8B chain 0 bytes queue RPC: 0 conn, 0 req/s, 0 us
2022-06-19 17:38:48 Syncing #173202 0xb0ac9b7e6 1812.82 blk/s 0.0 tx/s 0.0 Mgas/s 0+ 68 Qtd LI:#173270 1/25 peers 6 M8B chain 181 KIB queue RPC: 0 conn, 0 req/s, 0 us
2022-06-19 17:38:53 Syncing #182528 0x6f4e6a1e 1861.48 blk/s 2.2 tx/s 2.3 Mgas/s 0+ 13 Qtd LI:#182541 1/25 peers 5 M8B chain 19 KIB queue RPC: 0 conn, 0 req/s, 0 us
2022-06-19 17:38:58 Syncing #192614 0xb0b3a250e 2014.38 blk/s 0.0 tx/s 0.0 Mgas/s 0+ 87 Qtd LI:#192701 1/25 peers 5 M8B chain 130 KIB queue RPC: 0 conn, 0 req/s, 0 us
2022-06-19 17:39:03 Syncing #201591 0x3fe4000f 1792.02 blk/s 0.0 tx/s 0.0 Mgas/s 0+ 0 Qtd LI:#201591 1/25 peers 5 M8B chain 0 bytes queue RPC: 0 conn, 0 req/s, 0 us
2022-06-19 17:39:08 Syncing #211243 0xb0b8e2c9f 1927.70 blk/s 0.0 tx/s 0.0 Mgas/s 0+ 0 Qtd LI:#211243 1/25 peers 6 M8B chain 0 bytes queue RPC: 0 conn, 0 req/s, 0 us
2022-06-19 17:39:13 Syncing #221129 0xc1f853003 1984.62 blk/s 0.0 tx/s 0.0 Mgas/s 0+ 98 Qtd LI:#221276 1/25 peers 6 M8B chain 146 KIB queue RPC: 0 conn, 0 req/s, 0 us
2022-06-19 17:39:18 Syncing #231389 0x033482af 2022.16 blk/s 0.0 tx/s 0.0 Mgas/s 0+ 0 Qtd LI:#231389 1/25 peers 6 M8B chain 0 bytes queue RPC: 0 conn, 0 req/s, 0 us
```

Block Details		Block 138345	
Block Height	138345		
Timestamp	a year ago June-05-2021 01:17:50 AM +2 UTC		
Transactions	0 Transactions		
Miner	0x00D370a5C9771e3fF894160AED4961B4e8D2e066E		
Size	585 bytes		
Hash	0x71a40f3c219215255717fee06eeeb52ac1d6516ff89f6281d3eb47c6318ed215		
Parent Hash	0xf5c91992898f21033e883ff8d3bfdb9460068ff3cbb6ada3ad2e6c3e9f68b08		
Difficulty	340,282,366,920,938,463,463,374,607,431,768,211,454		
Total Difficulty	47,076,364,051,677,231,727,840,560,065,147,972,889,303,313		
Gas Used	0 0%		
Gas Limit	6,666,666		
Nonce	0x0000000000000000		

- Once started the Client will connect to the provided list of peers and start downloading all blocks
- It will store all transaction data in a local database that can be queried via the API that is exposed by the client
- Sync speed depends on the number of connected peers and internet speed
- The client can be paused and resumed later
- You can verify that you are downloading the correct blocks by comparing the hash of a specific block number via an Explorer like <https://exchangelog.minimal-gaia-x.eu/> or another node

Gaia-X Testnet JSON-RPC Methods



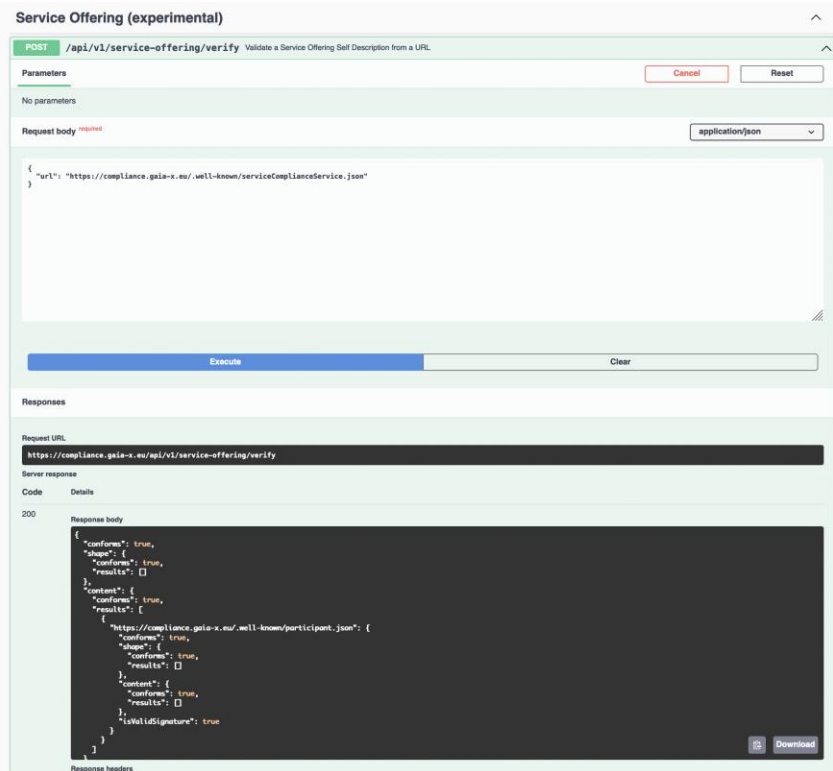
- OpenEthereum exposes a JSON-RPC interface that you can use to query the chain
- JSON-RPC is a stateless, light-weight remote procedure call (RPC) protocol
- OpenEthereum supports HTTP and WebSockets
- When requests are made that act on the state of Gaia-X Testnet, the last parameter determines the height of the block
- There are several datatypes that are passed over JSON. When encoding QUANTITIES (integers, numbers): encode as hex, prefix with “0x

```
18:33:34 main Δ curl --data '{"method":"eth_blockNumber","params":[],"id":1,"jsonrpc":"2.0"}'  
-H "Content-Type: application/json" -X POST localhost:8545  
{ "jsonrpc": "2.0", "result": "0x1ab321", "id": 1 }
```

Example call to query latest block using HTTP

```
curl --data '{"method":"eth_blockNumber","params":[],"id":1,"jsonrpc":"2.0"}' -H "Content-Type:  
application/json" -X POST localhost:8545
```

How to Create and Validate Self Descriptions?



Service Offering (experimental)

POST /api/v1/service-offering/verify Validate a Service Offering Self Description from a URL

Parameters

No parameters

Request body required application/json

```
{
  "url": "https://compliance.gaia-x.eu/well-known/serviceComplianceService.json"
}
```

Execute Clear

Responses

Request URL

https://compliance.gaia-x.eu/api/v1/service-offering/verify

Server response

Code	Details
200	<p>Response body</p> <pre>{ "conforms": true, "shape": { "conforms": true, "results": [] }, "content": { "conforms": true, "results": [{ "https://compliance.gaia-x.eu/well-known/participant.json": { "conforms": true, "shape": { "conforms": true, "results": [] }, "content": { "conforms": true, "results": [] }, "isValidSignature": true }] } } }</pre>

Response headers

- We are using the SD signer to create our Self Descriptions: <https://github.com/deltaDAO/self-description-signer>
- First, create your Self Description based on examples <https://gitlab.com/gaia-x/lab/compliance/gx-compliance/-/tree/main/src/tests/fixtures> or using the Trust Framework as reference at <https://gaia-x.gitlab.io/policy-rules-committee/trust-framework/>
- Next, adjust your Self Description based on the validation results from the Compliance Service
- Once done you can upload to a place of your choice and share your Self Description and others can verify it:
 - Using the Compliance Swagger UI: https://compliance.gaia-x.eu/docs/#/Participant/ParticipantController_verifyParticipant
 - Using a command line tool (or your favorite tool) like curl: `curl 'https://compliance.gaia-x.eu/api/v1/service-offering/verify' -X POST -H 'Content-Type: application/json' --data-raw '{ "url": "https://www.delta-dao.com/.well-known/serviceDLTvalidatorDeltaDAO.json" }'`

```
07:42:01 main Δ node index.js
Loaded ./config/self-description.json
Hashed canonized SD 43f98805a5eafab77467572e66dc1caad8afd285409e768e0513b52143b96a24
SD signed successfully (local)
Verification successful (local)
./output/1655703738115_self-signed_ServiceOfferingExperimental.json saved
./output/1655703738115_did.json saved

Checking Self Description with the Compliance Service...
SD signed successfully (compliance service)
Verification successful (compliance service)
./output/1655703738115_complete_ServiceOfferingExperimental.json saved
```

Using Self Descriptions to describe the Gaia-X Testnet



```
3  "@context": [
4    "http://www.w3.org/ns/shacl#",
5    "http://www.w3.org/2001/XMLSchema#",
6    "http://w3id.org/gaia-x/resource#",
7    "http://w3id.org/gaia-x/participant#",
8    "http://w3id.org/gaia-x/service-offering#"
9  ],
10 "@type": [
11   "VerifiableCredential",
12   "ServiceOfferingExperimental"
13 ],
14 "@id": "https://delta-dao.com/.well-known/serviceDLTvalidatorDeltaDAO.json",
15 "credentialSubject": {
16   "id": "https://delta-dao.com/.well-known/serviceDLTvalidatorDeltaDAO.json",
17   "gx-service-offering:providedBy": {
18     "@value": "https://delta-dao.com/.well-known/participant.json",
19     "@type": "xsd:string"
20   },
21   "gx-service-offering:name": {
22     "@value": "Gaia-X Test Network Validator Node",
23     "@type": "xsd:string"
24   },
25   "gx-service-offering:description": [
26     {
27       "@value": "Gaia-X Test Network Validator Node",
28       "@type": "xsd:string"
29     }
30   ],
31   "gx-service-offering:chainID": {
32     "@value": "2021000",
33     "@type": "xsd:string"
34   },
35   "gx-service-offering:chainSymbol": {
36     "@value": "GX",
37     "@type": "xsd:string"
38   },
39   "gx-service-offering:rpcAddress": {
40     "@value": "https://rpc.gaiaxtestnet.oceanprotocol.com",
41     "@type": "xsd:anyURI"
42   },
43   "gx-service-offering:chainExplorer": {
44     "@value": "https://blockscout.gaiaxtestnet.oceanprotocol.com",
45     "@type": "xsd:anyURI"
46   },
47   "gx-service-offering:nodeClient": {
48     "@value": "OpenEthereum",
49     "@type": "xsd:anyURI"
50   },
51   "gx-service-offering:nodeClientVersion": {
52     "@value": "3.2.6",
53     "@type": "xsd:anyURI"
54   },
55 }
```

- Using the Experimental Service Offering to describe the validator nodes and network itself
- Validator nodes and network Self Descriptions as service SDs:
 - <https://www.delta-dao.com/.well-known/serviceDLTvalidatorBDB.json>
 - <https://www.delta-dao.com/.well-known/serviceDLTvalidatorDeltaDAO.json>
 - <https://www.delta-dao.com/.well-known/serviceDLTnetwork.json>
- Using Participant Self Descriptions to define the provider of the service:
 - <https://delta-dao.com/.well-known/participantBigchainDB.json>
 - <https://delta-dao.com/.well-known/participant.json>

Useful Resources



- **Compliance Service Documentation:** <https://gitlab.com/gaia-x/lab/compliance/gx-compliance>
- **Compliance Service Guide:** <https://compliance.gaia-x.eu/guide/>
- **Compliance Service:** <https://compliance.gaia-x.eu/docs/>
- **SD Signer:** <https://github.com/deltaDAO/self-description-signer>
- **Gaia-X Testnet Node setup:** <https://github.com/oceanByte/gaia-x-node-setup>
- **Gaia-X Faucet:** <https://faucet.gx.gaiaxtestnet.oceanprotocol.com/>
- **Client Documentation** <https://openethereum.github.io/index>
- **OpenEthereum Client Releases:** <https://github.com/openethereum/openethereum/releases>
- **Gaia-X Testnet Explorer:** <https://exchangelog.minimal-gaia-x.eu/>
- **Gaia-X Testnet Stats:** <https://stats.minimal-gaia-x.eu/>
- **JSON-RPC methods:** <https://openethereum.github.io/JSONRPC-eth-module>