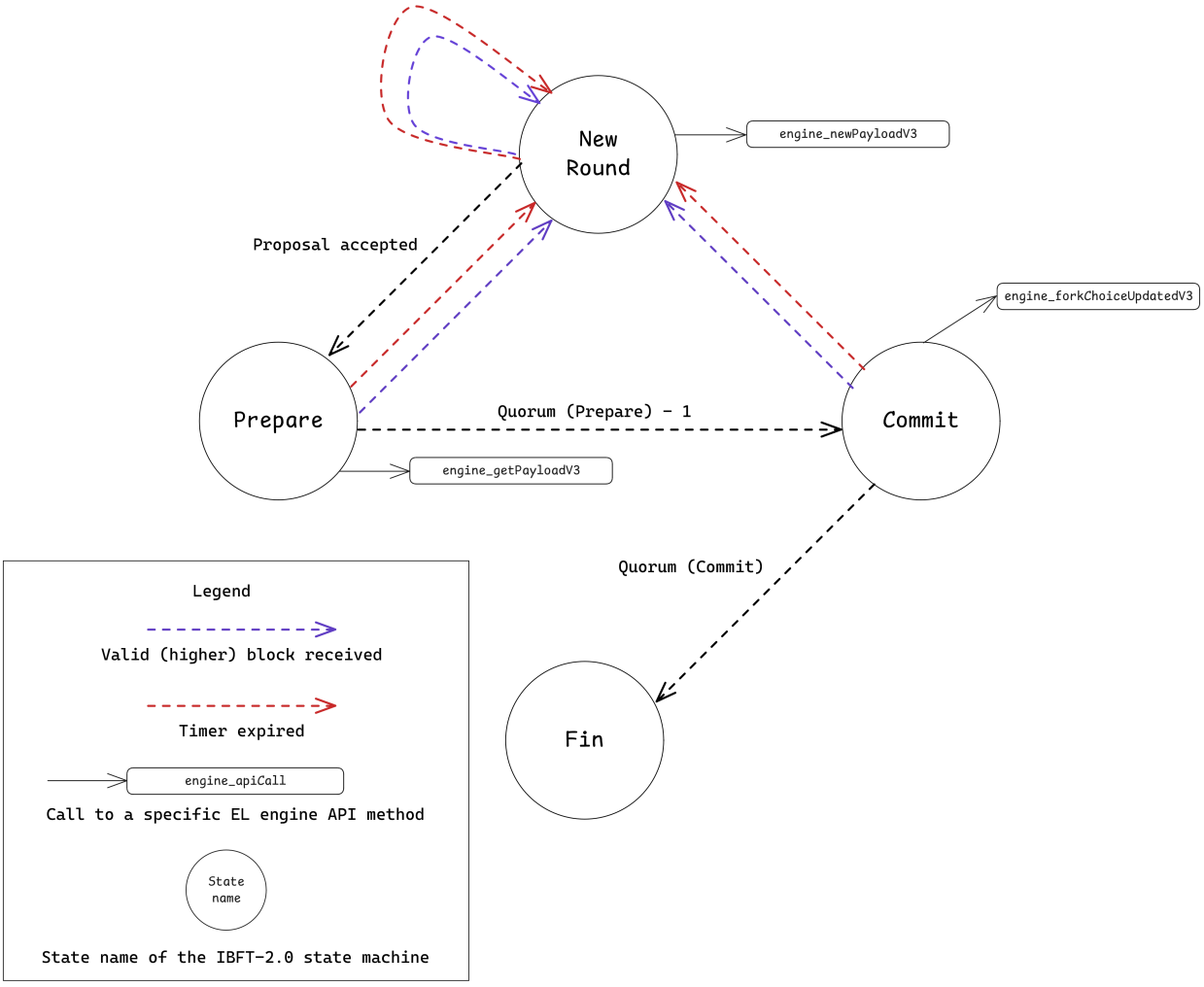
Getting started

Introduction



Nexus EVM network isp powered by a higly-performant IBFT-2.0 consensus client and a fork of go-ethereum as an execution client.

To find out more about Apex Fusion, visit the official website

Nexus is based on the work of the Polygon Edge EVM client, learn more about Polygon here.

If you would like to setup a local testnet for Nexus, for development purposes, please refer to Nexus Local repository

Ethereum compatibility

Nexus aims to be completely compatible with the open-source tools widely used in the Ethereum ecosystem.

This includes, but is not limited to:

- Support for Ethereum's JSON-RPC
- EVM support (Cancun EVM)

Differences to Ethereum

- Nexus does not support Blob transactions (EIP-4844)
- Nexus will always return `0x0` for the PREVRANDAO op-code
- Nexus introduces a single modification to the EIP-1559 spec, sending the portion of the gas fees that would get burned to the "miner"

Take a look at the "Planned Upgrades" for more information about planned network upgrades aiming at increasing Ethereum compatibility.

Architecture

Nexus is an blockchain client which uses EVM for execution and Istanbul Byzantine Fault Tolerance (EIP-650)) vote-elected PoA consensus mechanism powered by the IBFT-2.0 consensus algorithm with instant-finality.

Nexus is designed to be resource efficient and simple to setup.

Node Storage

Upon starting, Nexus will initialize two LevelDB databases, inside the specified `--data-dir`:

- `blockchain/`
  - Used for storing raw block data (including transactions)
  - Used when running JSON-RPC queries such as `eth_getBlockByNumber` or `eth_getTransactionByHash`
- `trie/`
  - The Merkle Patricia Trie holding the world state of the EVM.
  - On average it is ~10x in size than the `blockchain/` database

Apart from these two, Nexus also creates `consensus/metadata` and `consensus/snapshots` files for persisting the validator votes in the current IBFT epoch.

Node Networking

Nexus willl attempt to listen on these addresses and ports by default:

- `0.0.0.0:8545` - JSON-RPC
  - Used for interacting with the network thorugh Ethereum's JSON-RPC specification, used by the Execution client
- `0.0.0.0:30303` - DevP2P
  - Used as the P2P protocol of the execution clients
- `127.0.0.1:9632` - gRPC

    - Used for conducting administrative actions on the node, as well as casting validator votes
    - **Note the 127.0.0.1 being used as default listening IP because this is a privileged administrative API, posing a great security threat if left available publicly**
- `127.0.0.1:1478` - libp2p
    - Used for P2P connectivity between the nodes in the network
    - Should be made available publicly for common setups
- (optional) `0.0.0.0:5001` - Prometheus
    - Used for P2P connectivity between the nodes in the network
    - Should be made available publicly for common setups

Roles in the network

Although there are no constraints in place to how a node in the Nexus network may be utilized, it is practical to reason about five common roles a node might have in the network:

- RPC nodes
- Archive nodes
- Validator nodes
- Bootnodes

RPC Nodes

> *"If you're providing a public JSON-RPC node, make sure to rate limit based on IP or API key through 3rd party softwarre."*

Nodes usually made available for public network access or for exclusive access by a state indexing software.

> *"Tip: Caching JSON-RPC responses and configuring a load-balancer with multiple Nexus RPC node as upstreams is advised. Open-source projects like*

> *dschackle can be used for such purposes."*

Archive nodes

Node used for online backups to be restored later in other parts of your infrastructure to prevent time-consuming block syncing.

Not to be confused with Ethereum's concept of Archive nodes signifying the state retention policy.

Validator nodes

Nodes that secure the network by attesting to valid blocks proposed in the consensus and by proposing new blocks.

Nexus node automatically detects whether it bears the identity of an active validator and starts participating in the consensus, no manual configuration is needed - other than validator election.

Bootnodes

Nodes that are well-connected to the rest of the network, serving as coordination points exchanging other nodes' connection capabilities.

To connect to the Nexus network you must specify at least one node to be used as your bootnode.

Data Retention

Nexus only supports running in archive node operation, storing historical data on a per-block basis. This means that querying balances or smart contract state at a specific would be cheap, at the expense of disk space.

Usage

Instructions

Installing Nexus

Please refer to the installation method more applicable to you.

Our recommendation is to use the pre-built Docker Image.

Only Linux / AMD64 usage is tested and documented.

Docker Image

Official Linux / AMD64 Docker images are hosted on GitHub:

- Nexus IBFT-2.0 Consensus Client: GitHub.
- Nexus Execution Client: GitHub

```
docker pull ghcr.io/route3/nexus-beacon:latest
docker pull ghcr.io/route3/nexus-geth:latest
```

Building from source

> *"Prior to continuing, make sure you have Go >=1.18 installed and properly configured."*

```
git clone https://github.com/Apex-Fusion/nexus.git
cd nexus/go build -o nexus main.go
sudo mv nexus  /usr/local/bin
```

Running with Docker

The below command will run (and restart upon failure) a docker container running the Nexus node, with common data-dir mountpoints and forwarded ports.

```
docker run -d \
--restart always \
-v /mnt/data:/data \
-p 8545:8545 \
-p 1478:1478 \
```

```
ghcr.io/Apex-Fusion/nexus:latest \
--data-dir /data \
--chain data/genesis.json \
--jsonrpc 127.0.0.1:8545 \
--libp2p 0.0.0.0:1478
```

Running with Systemd

If you have compiled Nexus yourself, we advise you to manage it as a Systemnd service, an example configuration is provided below:

```
[Unit]
Description=Nexus Server
After=network.target
StartLimitIntervalSec=0

[Service]
Type=simple
Restart=always
RestartSec=10
User=nexus
ExecStart=/usr/local/bin/nexus server --config /etc/nexus/config.yaml

[Install]
WantedBy=multi-user.target
```

System Requirements

We recommend running blockchain nodes on bare-metal hardware.

Type	Minimum	Optimal	Influenced by
CPU	2 cores	8 cores	- Number of JSON-RPC queries - Gas consumption in the network

Type	Minimum	Optimal	Influenced by
RAM	2 GB	16 GB	- Size of the blockchain state - Size of the transaction pool
Disk	- 10 GB root partition - 30 GB data partition Use LVM for future disk extension	- 15 GB root patition - 90 GB data partition Use LVM for future disk extension	- Size of the blockchain state

Running a validator

> *“Please read "Getting Started" and "Running a Node" first.”*

Server configuration

Logging

The logs outputted by the Nexus node should:

- be sent to an external data store with indexing and searching capabilities
- have a log retention period of 30 days

If this is your first time setting up a validator, we recommend to start the node with the `--log-level=DEBUG` option to be able to quickly debug any issues you might face.

> *“The `--log-level=DEBUG` will make the node's log output be as verbose as possible. Debug logs will drastically increase the size of the log file which must be taken into account when setting up a log rotation solution.”*

Log Rotation

If written to disk, log files need to be rotated on a daily basis.

We suggest using `logrotate` with the following configuration:

```
{
        rotate 7
        daily
        missingok
        notifempty
        compress
}
```

Data Storage

The `--data-dir` folder containing the entire blockchain state should be mounted on a dedicated disk / volume allowing for automatic disk backups, volume extension and optionally mounting the disk/volume to another instance in case of failure.

Linux Hardening

> *"NOTE: Make sure to increase the open-file limit for the user running the `nexus` process."*

Use industry standard practices of a restrictive firewall and Linux capability / ownership set up. Make sure to enable OS security patches.

System Monitoring

Operators need to setup some kind of system metrics monitor, (e.g. Telegraf + InfluxDB + Grafana or a 3rd party SaaS).

Metrics that need to be monitored and that need to have alarm notifications setup:

Metric Name	Alarm threshold
CPU usage	> 90% for more than 5 minutes
RAM utilization)	> 90% for more than 5 minutes
Disk utilization	> 80%

Managing Secrets

Nexus has two types of private keys that it directly manages:

- Private key used for identifying with the consensus algorithm
- Private key used for encryption and authentication by libp2p

Consensus Key

> *"**DANGER:** The consensus private key is unique and secret to all nodes. The key is not to be shared with anyone!"*

The private key file mentioned as the *consensus private key* is also referred to as the **validator private key**. This private key is used when the node is acting as a validator in the network and needs to sign new data.

The private key file is by default located in **`<data-dir>/consensus/validator.key`**.

Libp2p Key

> *"**DANGER:** The libp2p private key is unique and secret to all nodes. The key is not to be shared with anyone!"*

The private key file mentioned for networking is used by libp2p to generate the corresponding PeerID, and allow the node to participate in the network.

It is located in **`keystore/libp2p.key`**.

Key Format

The private keys are stored in Base64 format, so they can be human-readable and portable.

```
# Example private key
0802122068a1bdb1c8af5333e58fe586bc0e9fc7aff882da82affb678aef5d9a2b910
```

All private key files generated and used inside the Nexus are relying on ECDSA with the curve secp256k1, as used in Ethereum and Bitcoin.

As the curve is non-standard, it cannot be encoded and stored in any standardized PEM format. Importing keys that don't conform to this key type is not supported.

Tutorial

> *"**NOTE:** This tutorial for generating secrets is recommended just for running validator nodes so that you can share the validator address to be voted-in the consensus as a validator. If secrets are missing in the expected location, the `**nexus server**` command will automatically generate them."*

First step is to initialize the secrets in your specified `**<data-dir>**`:

```
nexus secrets init --data-dir <data-dir>
```

Each of these commands will print the public validator address, and the node ID.

You will use the outputted public validator address to share with other validators for voting you in.

The secrets public key output can be retrieved again, if needed:

```
nexus secrets output --data-dir <data-dir>
```

Setup Procedure

Following is the desired flow of the setup procedure for a Nexus validator, for exact values consult the [Testnet datasheet]

- ☐ Read through this documentation
- ☐ Prepare the system configuration according to suggestions

- ☐ Install the latest release of Nexus (both execution and consensus clients)
- ☐ Generate and store validator secrets using the `nexus secrets init` CLI command (`nexus secrets init --help` is available)
- ☐ Take note of `NodeID` and `Public key (address)` values returned by `secrets init`
- ☐ Download both `genesis.json` files from the documentastion
- ☐ Create a Nexus service (`systemd` or similar) and properly setup paths
- ☐ Start the Nexus EL service, query `admin.peers` to check for P2P connectivity
- ☐ Start the Nexus server by starting the service (e.g. `systemctl start nexus`)
- ☐ Check the `nexus` log output and make sure the blocks are being generated and that there are no `[ERROR]` logs in the console
- ☐ Check the node responsiveness and status by issuing gRPC and JSON-RPC commands
- ☐ Expose only the JSON-RPC of the exeuciton client for public access

Querying node information

A functioning node is required in order to query any kind of operator information.

You can use the built-in CLI commands and get meaningful information about your node - no log sifting required.

> *"**Note:** If your node isn't running on `127.0.0.1:8545` you should add a flag `--grpc-address <address:port>` to the commands listed in this document."*

Peer Information

Peers List

To get a complete list of connected peers (including the running node itself), run the following command:

```
nexus peers list
```

Peer Status

For the status of a specific peer, run:

```
nexus peers status --peer-id <address>
```

With the *address* parameter being the libp2p address of the peer.

Consensus Information

Snapshots

Running the following command returns the most recent validator set snapshot:

```
nexus ibft snapshot
```

To query the snapshot at a specific height (block number), the operator can run:

```
nexus ibft snapshot --num <block-number>
```

Candidates

To get the latest info on validator candidates, the operator can run:

```
nexus ibft candidates
```

Transaction Pool

To find the current number of transactions in the transaction pool, the operator can run:

```
nexus txpool status
```
