faq.md:

---
sidebarlabel: FAQ
sidebarposition: 7

:::warning
This article requires a revision.
:::

FAQ

How do I upgrade my node?

Upgrading Nethermind for Docker users

Pull the latest version of Nethermind using the latest tag. You can find specific tag
at Docker Hub.

sh
docker pull nethermind/nethermind:latest

or pull with docker-compose if the nethermind/nethermind:latest base image is specified

sh
docker-compose pull

Upgrading Nethermind for Systemd users

Simply download the latest Nethermind package either
from https://downloads.nethermind.io
or https://github.com/NethermindEth/nethermind/releases and make
sure that the package is extracted in the WorkingDirectory path defined in your systemd service. Make sure to stop
the service before the upgrade and start it after.

systemd
WorkingDirectory=/home/nethermind/nethermind

Upgrading Nethermind when running as a background process

Download the latest Nethermind package either from
https://downloads.nethermind.io
or https://github.com/NethermindEth/nethermind/releases and
extract the package in the folder you currently use for running Nethermind. Make sure the service is down before the
update. Make sure to stop the service before the upgrade and start it after.

What is the minimum viable config to serve ETH2 validator requests?

This config downloads a minimal amount of bodies and receipts to be able to serve ETH2 validator requests since the
deposit contract deployment. It also enables JSON RPC (important - make sure that you do not open firewall to the
outside world!)

sh
nethermind --config mainnet --Init.BaseDbPath /your/db/path

If you have any issues, please reach out to us on Discord:
https://discord.gg/X539yhn.

Can I disable logging to file?

You can find more details on the logging config page here

Can I disable logging of JSON RPC calls?

You can find more details on the logging config page here

How can I configure a validator?

You can find more details on running validators here.

My network bandwidth is used up by the Nethermind node

Try changing the config to a lower number (--Network.ActivePeersMaxCount 25)

Is my node synced?

Your node is synced when it shows log lines starting with:

Processed ...

And the block numbers shown are at the head of the chain.

Also, to check if your node is synced, you may execute ethsyncing call check its result

text
curl -X POST --data '{"jsonrpc":"2.0","method":"ethsyncing","params":[],"id":1}' localhost:8545

If the result shows false it means that your node is synced

text
{"jsonrpc":"2.0","result":false,"id":1}

:::info
In blockchain, any node can never be 100% sure it is synced because there is no central source of truth - so your node
is generally not able to tell you that it is synced but, it is able to tell you that it believes it is synced based on
what it knows from the peers that it talks to).
:::

I want to run two or more nodes on one machine. How can I achieve this?

You have to configure a few ports used by Nethermind.

Settings	Default Value	Comment
JsonRpc.EnginePort	8551	Remember to change it on your Consensus client.
JsonRpc.Port	8545	Only needed if you are using standard

```
JsonRpc                                |
| JsonRpc.WebSocketsPort | 8545          | By default same as the JsonRpc port.
Only relevant when you use WebSockets |
| Network.P2PPort        | 30303         |
|
| Network.DiscoveryPort  | 30303         |
|
```

The example of parameters that you have to pass to your second node when the first has been running with default settings:\
--JsonRpc.EnginePort 8552 --JsonRpc.Port 8546 --Network.P2PPort 30304 --Network.DiscoveryPort 30304

troubleshooting.md:

---
title: Troubleshooting
sidebarposition: 8
---

import Tabs from "@theme/Tabs";
import TabItem from "@theme/TabItem";

File descriptor limits

In some cases, file descriptor limits may cause errors like "Too many open files". To solve that, see the instructions for your platform below.

<Tabs groupId="os">
<TabItem value="linux" label="Linux">

To increase the limits for the user running Nethermind (given the process name of nethermind), run:

bash
sudo echo "nethermind soft nofile 100000" >
/etc/security/limits.d/nethermind.conf
sudo echo "nethermind hard nofile 100000" >>
/etc/security/limits.d/nethermind.conf

</TabItem>
<TabItem value="macos" label="macOS">

To increase the limits, run:

bash
ulimit -n 10000

If you run into issues with the above command, see the workaround.

Note that the changes above are temporary and will be reset after the system reboot. To make them permanent, you can add them to your /.bashrc or /.bashprofile shell configuration file.

</TabItem>
</Tabs>

Database corruption issues

Database corruption is one of the issues that happen now and then; it has many

possible causes among them:

- Hardware failures: disk failures, memory errors, hardware overheating, etc.
- Power cuts and abrupt shutdowns

There's no shortcut in such situations, and resyncing Nethermind from scratch is the recommended remedy.

TrieNodeException errors

If Nethermind reports Nethermind.Trie.TrieNodeException or Nethermind.Trie.MissingTrieNodeException errors, that usually indicates database corruption or missing data. The following steps may help:

- If the node is still syncing, wait until it has been fully synced and observe for errors
- Restart the node and observe for errors
- Update to the latest version of Nethermind
- If the above steps fail, a resync may be required

Issues with lock files

If Nethermind complains about the lock files, it perhaps because of one of the following:

- Another Nethermind process is running using the same database
- The database has not been appropriately closed on the last run.\
  In this case, run the following command from the Nethermind database directory:

```
bash
find . -type f -name 'LOCK' -delete
```

Block checksum mismatch

Sometimes Nethermind may fail with an error similar the following:

```
Corruption: block checksum mismatch: expected 2087346143, got 2983326672 in...
```

This tends to happen on XFS file systems under very high memory pressure. The issue can be mitigated by using the --Db.UseDirectIoForFlushAndCompactions true option although at the cost of performance.

However, quite often, this is because of memory module issues.

Plugin loading failure

If Nethermind fails to start with a message like Failed to load plugin..., this is most likely due to a missing or incompatible plugin. Reinstalling Nethermind usually fixes the issue.

:::tip
If you install Nethermind over an existing installation, remove the old installation first, particularly the plugins directory. Package managers do this automatically.
:::

building-from-source.md:

---
title: Building from source
sidebarlabel: Building from source
sidebarposition: 0

The Nethermind's source code can be obtained from our repository on GitHub:

bash
git clone --recursive https://github.com/nethermindeth/nethermind.git

There are two options buiding Nethermind from source code:

 Standalone binaries
 Docker image

Building standalone binaries

Prerequisites

To build Nethermind from source, install .NET SDK 8 or later.

Building

To build both the client and tests, run the following command from the project's root directory:

bash
dotnet build src/Nethermind/Nethermind.sln -c release

To simply run the client with a specific configuration without building tests, see below.

:::info
Before running the client or tests, ensure the platform-specific prerequisites are met.
:::

Running

Nethermind can be launched immediately without compiling explicitly (thus, the previous step can be skipped). The following command builds Nethermind if needed and runs it:

bash
cd src/Nethermind/Nethermind.Runner
dotnet run -c release -- -c mainnet

All Nethermind-specific parameters can be specified after --. For instance, the command above specifies the Mainnet configuration only.

The build artifacts can be found in the src/Nethermind/artifacts/bin/Nethermind.Runner/release directory. By default, the logs and database directories are located here as well.

For more info, see Running Nethermind.

Testing

There are two test suites â ¨ Nethermind and Ethereum Foundation. Tests can be run with the following commands (the
initial step of the build is not required):

bash
cd src/Nethermind

Run Nethermind tests
dotnet test Nethermind.sln -c release

Run Ethereum Foundation tests
dotnet test EthereumTests.sln -c release

Bulding Docker image

To build Nethermind Docker image, run the following command from the project's root directory:

bash
docker build -t nethermind .

For more info about running Docker containers,
see Installing Nethermind.

custom-analytic-tools.md:

---
title: Custom analytic tools
sidebarposition: 2

:::warning
This article requires a revision.
:::

Check plugins for some easy addition of analytical tools

:::info
Learn more about Plugins
:::

You can also read more about some useful interfaces below:

There are multiple extension points where you can add custom analytics to your Nethermind node if you know some C#.
Below you will find an example of using two very useful interfaces - IBlockVisitor and ITreeVisitor.

Just to execute the code I have added one new initialization step that invokes two custom verifiers that I have used for
calculating total supply in two different ways - by calculating mining rewards and by summing up all account balances:

```
    [RunnerStepDependencies(typeof(ReviewBlockTree))]
    public class RunCustomTools : IStep
    {
        private readonly EthereumRunnerContext context;

        public RunCustomTools(EthereumRunnerContext context)
        {
```

```
            context = context;
        }

        public Task Execute(CancellationToken cancellationToken)
        {
            ILogger logger = context.LogManager.GetClassLogger();
            IInitConfig initConfig = context.Config<IInitConfig>();

            switch (initConfig.DiagnosticMode)
            {
                case DiagnosticMode.VerifySupply:
                {
                    logger.Info("Genesis supply:");
                    SupplyVerifier supplyVerifier = new SupplyVerifier(logger);
                    StateDb stateDb = new
StateDb(context.DbProvider.StateDb.Innermost);
                    StateDb codeDb = new
StateDb(context.DbProvider.StateDb.Innermost);
                    StateReader stateReader = new StateReader(stateDb, codeDb,
context.LogManager);
                    stateReader.RunTreeVisitor(supplyVerifier,
context.BlockTree!.Genesis.StateRoot);

                    Block head = context.BlockTree!.Head;
                    logger.Info($"Head ({head.Number}) block supply:");
                    supplyVerifier = new SupplyVerifier(logger);
                    stateReader.RunTreeVisitor(supplyVerifier, head.StateRoot);
                    break;
                }
                case DiagnosticMode.VerifyRewards:
                    context.BlockTree!.Accept(new
RewardsVerifier(context.LogManager), cancellationToken);
                    break;
            }

            return Task.CompletedTask;
        }
    }
```

Below you will see an example of using ITreeVisitor that allows to check all the blocks, including some of the discarded
branches if you wish so:

```
    public class RewardsVerifier : IBlockTreeVisitor
    {
        private ILogger logger;
        public bool PreventsAcceptingNewBlocks => true;
        public long StartLevelInclusive => 0;
        public long EndLevelExclusive => 10618000;

        private UInt256 genesisAllocations =
UInt256.Parse("72009990499480000000000000");
        private UInt256 uncles;
        private UInt256 blockRewards;

        public RewardsVerifier(ILogManager logManager)
        {
            logger = logManager.GetClassLogger();
        }

        private RewardCalculator rewardCalculator = new
RewardCalculator(MainnetSpecProvider.Instance);
```

```
        public Task<BlockVisitOutcome> VisitBlock(Block block, CancellationToken
cancellationToken)
        {
            BlockReward[] rewards = rewardCalculator.CalculateRewards(block);
            for (int i = 0; i < rewards.Length; i++)
            {
                if (rewards[i].RewardType == BlockRewardType.Uncle)
                {
                    uncles += rewards[i].Value;
                }
                else
                {
                    blockRewards += rewards[i].Value;
                }
            }

            logger.Info($"Visiting block {block.Number}, total supply is
(genesis + miner rewards + uncle rewards) | {genesisAllocations} +
{blockRewards} + {uncles}");
            return Task.FromResult(BlockVisitOutcome.None);
        }

        public Task<LevelVisitOutcome> VisitLevelStart(ChainLevelInfo
chainLevelInfo, CancellationToken cancellationToken)
            => Task.FromResult(LevelVisitOutcome.None);

        public Task<bool> VisitMissing(Keccak hash, CancellationToken
cancellationToken)
            => Task.FromResult(true);

        public Task<bool> VisitHeader(BlockHeader header, CancellationToken
cancellationToken)
            => Task.FromResult(true);

        public Task<LevelVisitOutcome> VisitLevelEnd(CancellationToken
cancellationToken)
            => Task.FromResult(LevelVisitOutcome.None);
    }
```

And here you will find an example of a tree visitor that sums up all the account balances:

```
    public class SupplyVerifier : ITreeVisitor
    {
        private readonly ILogger logger;
        private UInt256 balance = UInt256.Zero;
        private int accountsVisited;

        public SupplyVerifier(ILogger logger)
        {
            logger = logger;
        }

        public bool ShouldVisit(Keccak nextNode) { return true; }

        public void VisitTree(Keccak rootHash, TrieVisitContext
trieVisitContext) { }

        public void VisitMissingNode(Keccak nodeHash, TrieVisitContext
trieVisitContext) { }
```

```
        public void VisitBranch(TrieNode node, TrieVisitContext
trieVisitContext) { }

        public void VisitExtension(TrieNode node, TrieVisitContext
trieVisitContext) { }

        public void VisitLeaf(TrieNode node, TrieVisitContext trieVisitContext,
byte[] value = null)
        {
            if (trieVisitContext.IsStorage)
            {
                return;
            }

            AccountDecoder accountDecoder = new AccountDecoder();
            Account account = accountDecoder.Decode(node.Value.AsRlpStream());
            balance += account.Balance;
            accountsVisited++;

            logger.Info($"Balance after visiting {accountsVisited}: {balance}");
        }

        public void VisitCode(Keccak codeHash, TrieVisitContext
trieVisitContext) { }
    }
```

```
# plugins.md:

---
title: Plugins
sidebarposition: 1
---

:::warning
This article requires a revision.
:::

Nethermind plugins is a powerful way of extending your local node capabilities.

\(see also an article
here: https://medium.com/nethermind-eth/writing-your-first-nethermind-plugin-
a9e04d81cf59\)

Plugins that you can write:

| Plugin Type         | What can it be used for?
|
|:--------------------|:-------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------|
| RPC                 | Adding additional RPC modules to the client that have
full access to the internal Nethermind APIs and can extend capabilities of the
node when integrating with your infrastructure / systems. |
| Block Tree Visitors | Code allowing you to analyze entire block tree from
genesis to the head block and execute aggregated calculations and checks.
|
| Devp2p              | Allows you to create additional devp2p network protocol
for your nodes to communicate over TCP/IP. You can also build custom products
that will run attached to Nethermind nodes.              |
| State Visitors      | Allow you to run aggregated analysis on the entire raw
format state \(or just some accounts storages\).
|
```

Config	You can add additional configuration categories to our config files and then use them in env variables, json files or command line to configure behaviour of your plugins.
TxPool	TxPool behaviours and listeners.
Tracers	Custom, powerful EVM tracers capable of extracting elements of EVM execution in real time.
CLI	Additional modules for Nethermind CLI that can allow you build some quick scratchpad style JavaScript based behaviors.

Note: When writing a plugin be carefull about exceptions you throw. Especially if you are hooking up event handlers
on some core objects like BlockProcessor or BlockTree. Those exceptions can bring the node down. This is by design. Its
responsibility of plugin writer to correctly handle those exceptions.

How to build a plugin? We included an example inside the Nethermind.Analytics plugin:

RPC Plugin example:

```
csharp
    [RpcModule(ModuleType.Clique)]
    public interface IAnalyticsModule : IModule
    {
        [JsonRpcMethod(Description = "Retrieves ETH supply counted from state.", IsImplemented = true)]
        ResultWrapper<UInt256> analyticsverifySupply();

        [JsonRpcMethod(Description = "Retrieves ETH supply counted from rewards.", IsImplemented = true)]
        ResultWrapper<UInt256> analyticsverifyRewards();
    }
```

CLI Plugin example:

```
csharp
[CliModule("analytics")]
public class AnalyticsCliModule : CliModuleBase
{
    [CliFunction("analytics", "verifySupply")]
    public string VerifySupply()
    {
        return NodeManager.Post<string>("analyticsverifySupply").Result;
    }

    [CliFunction("analytics", "verifyRewards")]
    public string VerifyRewards()
    {
        return NodeManager.Post<string>("analyticsverifyRewards").Result;
    }

    public AnalyticsCliModule(ICliEngine cliEngine, INodeManager nodeManager)
        : base(cliEngine, nodeManager) { }
}
```

Block Tree Visitor Plugin example:

```
csharp
    public class RewardsVerifier : IBlockTreeVisitor
    {
        private ILogger logger;
        public bool PreventsAcceptingNewBlocks => true;
        public long StartLevelInclusive => 0;
        public long EndLevelExclusive { get; }

        private UInt256 genesisAllocations =
UInt256.Parse("72009990499480000000000000");
        private UInt256 uncles;

        public UInt256 BlockRewards { get; private set; }

        public RewardsVerifier(ILogManager logManager, long endLevelExclusive)
        {
            logger = logManager.GetClassLogger();
            EndLevelExclusive = endLevelExclusive;
            BlockRewards = genesisAllocations;
        }

        private RewardCalculator rewardCalculator = new
RewardCalculator(MainnetSpecProvider.Instance);

        public Task<BlockVisitOutcome> VisitBlock(Block block, CancellationToken
cancellationToken)
        {
            BlockReward[] rewards = rewardCalculator.CalculateRewards(block);
            for (int i = 0; i < rewards.Length; i++)
            {
                if (rewards[i].RewardType == BlockRewardType.Uncle)
                {
                    uncles += rewards[i].Value;
                }
                else
                {
                    BlockRewards += rewards[i].Value;
                }
            }

            logger.Info($"Visiting block {block.Number}, total supply is
(genesis + miner rewards + uncle rewards) | {genesisAllocations} +
{BlockRewards} + {uncles}");
            return Task.FromResult(BlockVisitOutcome.None);
        }

        public Task<LevelVisitOutcome> VisitLevelStart(ChainLevelInfo
chainLevelInfo, CancellationToken cancellationToken)
            => Task.FromResult(LevelVisitOutcome.None);

        public Task<bool> VisitMissing(Keccak hash, CancellationToken
cancellationToken)
            => Task.FromResult(true);

        public Task<HeaderVisitOutcome> VisitHeader(BlockHeader header,
CancellationToken cancellationToken)
            => Task.FromResult(HeaderVisitOutcome.None);

        public Task<LevelVisitOutcome> VisitLevelEnd(CancellationToken
cancellationToken)
            => Task.FromResult(LevelVisitOutcome.None);
    }
```

Config plugin example:

csharp
```
public class AnalyticsConfig : IAnalyticsConfig
{
    public bool PluginsEnabled { get; set; }
    public bool StreamTransactions { get; set; }
    public bool StreamBlocks { get; set; }
    public bool LogPublishedData { get; set; }
}
```

State Tree Visitor example:

csharp
```
public class SupplyVerifier : ITreeVisitor
{
    private readonly ILogger logger;
    private HashSet<Keccak> ignoreThisOne = new HashSet<Keccak>();
    private int accountsVisited;
    private int nodesVisited;

    public SupplyVerifier(ILogger logger)
    {
        logger = logger;
    }

    public UInt256 Balance { get; set; } = UInt256.Zero;

    public bool ShouldVisit(Keccak nextNode)
    {
        if (ignoreThisOne.Count > 16)
        {
            logger.Warn($"Ignore count leak -> {ignoreThisOne.Count}");
        }

        if (ignoreThisOne.Contains(nextNode))
        {
            ignoreThisOne.Remove(nextNode);
            return false;
        }

        return true;
    }

    public void VisitTree(Keccak rootHash, TrieVisitContext trieVisitContext)
    {
    }

    public void VisitMissingNode(Keccak nodeHash, TrieVisitContext
trieVisitContext)
    {
        logger.Warn($"Missing node {nodeHash}");
    }

    public void VisitBranch(TrieNode node, TrieVisitContext trieVisitContext)
    {
        logger.Info($"Balance after visiting {accountsVisited} accounts and
{nodesVisited} nodes: {Balance}");
        nodesVisited++;

        if (trieVisitContext.IsStorage)
        {
            for (int i = 0; i < 16; i++)
```

```
            {
                Keccak childHash = node.GetChildHash(i);
                if (childHash != null)
                {
                    ignoreThisOne.Add(childHash);
                }
            }
        }
    }

    public void VisitExtension(TrieNode node, TrieVisitContext trieVisitContext)
    {
        nodesVisited++;
        if (trieVisitContext.IsStorage)
        {
            ignoreThisOne.Add(node.GetChildHash(0));
        }
    }

    public void VisitLeaf(TrieNode node, TrieVisitContext trieVisitContext,
byte[] value = null)
    {
        nodesVisited++;

        if (trieVisitContext.IsStorage)
        {
            return;
        }

        AccountDecoder accountDecoder = new AccountDecoder();
        Account account = accountDecoder.Decode(node.Value.AsRlpStream());
        Balance += account.Balance;
        accountsVisited++;

        logger.Info($"Balance after visiting {accountsVisited} accounts and
{nodesVisited} nodes: {Balance}");
    }

    public void VisitCode(Keccak codeHash, TrieVisitContext trieVisitContext)
    {
        nodesVisited++;
    }
}
```

```
# configuration.md:

---
title: Configuration
sidebarposition: 1
tocmaxheadinglevel: 4
---

Nethermind is highly configurable. There are 3 ways of configuring it, listed by
priority:

- Command line options (aka arguments or flags)
- Environment variables
- Configuration file

:::note
Given the above priority list, an option defined in a more priority way
```

overrides the same option defined elsewhere if any.
:::

Command line options

The full list of command line options can be displayed by running:

bash
nethermind -h

Below is the list of the basic options followed by an exhaustive list of options by namespace.

:::warning
The command line options are case-sensitive and can be defined only once unless stated otherwise.
:::

Basic options

- -d, --baseDbPath <path>

  The path to the Nethermind database directory. Defaults to db.

- -c, --config <value>

  The path to the configuration file or the name (without extension) of any of the configuration files in the configuration directory. Defaults to mainnet.

  <details>
  <summary>Available configurations</summary>

  Nethermind provides the following pre-built configurations named as the networks they are for. Their respective versions for archive nodes are suffixed archive.

  - base-mainnet base-mainnetarchive
  - base-sepolia base-sepoliaarchive
  - chiado chiadoarchive
  - energyweb energywebarchive
  - gnosis gnosisarchive
  - holesky holeskyarchive
  - mainnet mainnetarchive
  - op-mainnet op-mainnetarchive
  - op-sepolia op-sepoliaarchive
  - sepolia sepoliaarchive
  - volta voltaarchive

  </details>

- -cd, --configsDirectory <path>

  The path to the configuration files directory. Defaults to configs.

- -dd, --datadir <path>

  The path to the Nethermind data directory. Defaults to Nethermind's current directory.

  :::warning
  The absolute paths set by Init.BaseDbPath, Init.LogDirectory, or

KeyStore.KeyStoreDirectory options in a configuration file are not overridden by --datadir.
  :::

- -?, -h, --help

  Displays the full list of available command line options.

- -l, --log <level>

  Log level (severity). Allowed values: TRACE DEBUG INFO WARN ERROR OFF. Defaults to INFO.

- -lcs, --loggerConfigSource <path>

  The path to the NLog configuration file. Defaults to NLog.config.

- -pd, --pluginsDirectory <path>

  The path to the Nethermind plugins directory. Defaults to plugins.

- -v, --version

  Displays the Nethermind version info.

Options by namespaces

<!--[start autogen]-->

<details>
<summary className="nd-details-heading">

Aura

</summary>
<p>

- --Aura.AllowAuRaPrivateChains <value> NETHERMINDAURACONFIGALLOWAURAPRIVATECHAINS

  Whether to allow private Aura-based chains only. Do not use with existing Aura-based chains. Allowed values: true false. Defaults to false.

- --Aura.ForceSealing <value> NETHERMINDAURACONFIGFORCESEALING

  Whether to seal empty blocks if mining. Allowed values: true false. Defaults to true.

- --Aura.Minimum2MlnGasPerBlockWhenUsingBlockGasLimitContract <value> NETHERMINDAURACONFIGMINIMUM2MLNGASPERBLOCKWHENUSINGBLOCKGASLIMITCONTRACT

  Whether to use 2M gas if the contract returns less than that when using BlockGasLimitContractTransitions. Allowed values: true false. Defaults to false.

- --Aura.TxPriorityConfigFilePath <value> NETHERMINDAURACONFIGTXPRIORITYCONFIGFILEPATH

  The path to the transaction priority rules file to use when selecting transactions from the transaction pool. Defaults to null.

- --Aura.TxPriorityContractAddress <value> NETHERMINDAURACONFIGTXPRIORITYCONTRACTADDRESS

  The address of the transaction priority contract to use when selecting

transactions from the transaction pool. Defaults to null.

</details>

<details>
<summary className="nd-details-heading">

Blocks

</summary>

- --Blocks.ExtraData <value> NETHERMINDBLOCKSCONFIGEXTRADATA

  The block header extra data up to 32 bytes in length. Defaults to Nethermind.

- --Blocks.MinGasPrice <value> NETHERMINDBLOCKSCONFIGMINGASPRICE

  The minimum gas premium (or the gas price before the London hard fork) for transactions accepted by the block producer. Defaults to 1.

- --Blocks.PreWarmStateOnBlockProcessing <value> NETHERMINDBLOCKSCONFIGPREWARMSTATEONBLOCKPROCESSING

  Try to pre-warm the state when processing blocks. Can lead to 2x speedup in main loop block processing. Allowed values: true false. Defaults to True.

- --Blocks.RandomizedBlocks <value> NETHERMINDBLOCKSCONFIGRANDOMIZEDBLOCKS

  Whether to change the difficulty of the block randomly within the constraints. Used in NethDev only. Allowed values: true false. Defaults to false.

- --Blocks.SecondsPerSlot <value> NETHERMINDBLOCKSCONFIGSECONDSPERSLOT

  The block time slot, in seconds. Defaults to 12.

- --Blocks.TargetBlockGasLimit <value> NETHERMINDBLOCKSCONFIGTARGETBLOCKGASLIMIT

  The block gas limit that the block producer should try to reach in the fastest possible way based on the protocol rules. If not specified, then the block producer should follow others. Defaults to null.

</details>

<details>
<summary className="nd-details-heading">

Bloom

</summary>

- --Bloom.Index <value> NETHERMINDBLOOMCONFIGINDEX

  Whether to use the Bloom index. The Bloom index speeds up the RPC log searches. Allowed values: true false. Defaults to true.

- --Bloom.IndexLevelBucketSizes <value> NETHERMINDBLOOMCONFIGINDEXLEVELBUCKETSIZES

  An array of multipliers for index levels. Can be tweaked per chain to boost performance. Defaults to [4, 8, 8].

- --Bloom.Migration <value> NETHERMINDBLOOMCONFIGMIGRATION

  Whether to migrate the previously downloaded blocks to the Bloom index. Allowed values: true false. Defaults to false.

- --Bloom.MigrationStatistics <value> NETHERMINDBLOOMCONFIGMIGRATIONSTATISTICS

  Whether the migration statistics should be calculated and output. Allowed values: true false. Defaults to false.

</details>

<details>
<summary className="nd-details-heading">

EthStats

</summary>

- --EthStats.Contact <value> NETHERMINDETHSTATSCONFIGCONTACT

  The node owner contact details displayed on Ethstats. Defaults to hello@nethermind.io.

- --EthStats.Enabled <value> NETHERMINDETHSTATSCONFIGENABLED

  Whether to use Ethstats publishing. Allowed values: true false. Defaults to false.

- --EthStats.Name <value> NETHERMINDETHSTATSCONFIGNAME

  The node name displayed on Ethstats. Defaults to Nethermind.

- --EthStats.Secret <value> NETHERMINDETHSTATSCONFIGSECRET

  The Ethstats secret. Defaults to secret.

- --EthStats.SendInterval <value> NETHERMINDETHSTATSCONFIGSENDINTERVAL

  The stats update interval, in seconds. Defaults to 15.

- --EthStats.Server <value> NETHERMINDETHSTATSCONFIGSERVER

  The Ethstats server URL. Defaults to ws://localhost:3000/api.

</details>

<details>
<summary className="nd-details-heading">

HealthChecks

</summary>

- --HealthChecks.Enabled <value> NETHERMINDHEALTHCHECKSCONFIGENABLED

  Whether to enable the health check. Allowed values: true false. Defaults to false.

- --HealthChecks.LowStorageCheckAwaitOnStartup <value>
NETHERMINDHEALTHCHECKSCONFIGLOWSTORAGECHECKAWAITONSTARTUP

  Whether to check for low disk space on startup and suspend until enough space is available. Allowed values: true false. Defaults to false.

- --HealthChecks.LowStorageSpaceShutdownThreshold <value>
NETHERMINDHEALTHCHECKSCONFIGLOWSTORAGESPACESHUTDOWNTHRESHOLD

  The percentage of available disk space below which Nethermind shuts down. 0 to disable. Defaults to 1.

- --HealthChecks.LowStorageSpaceWarningThreshold <value>
NETHERMINDHEALTHCHECKSCONFIGLOWSTORAGESPACEWARNINGTHRESHOLD

  The percentage of available disk space below which a warning is displayed. 0 to disable. Defaults to 5.

- --HealthChecks.MaxIntervalClRequestTime <value>
NETHERMINDHEALTHCHECKSCONFIGMAXINTERVALCLREQUESTTIME

  The max request interval, in seconds, in which the consensus client is assumed healthy. Defaults to 300.

- --HealthChecks.MaxIntervalWithoutProcessedBlock <value>
NETHERMINDHEALTHCHECKSCONFIGMAXINTERVALWITHOUTPROCESSEDBLOCK

  The max interval, in seconds, in which the block processing is assumed healthy. Defaults to null.

- --HealthChecks.MaxIntervalWithoutProducedBlock <value>
NETHERMINDHEALTHCHECKSCONFIGMAXINTERVALWITHOUTPRODUCEDBLOCK

  The max interval, in seconds, in which the block production is assumed healthy. Defaults to null.

- --HealthChecks.PollingInterval <value>
NETHERMINDHEALTHCHECKSCONFIGPOLLINGINTERVAL

  The health check updates polling interval, in seconds. Defaults to 5.

- --HealthChecks.Slug <value> NETHERMINDHEALTHCHECKSCONFIGSLUG

  The URL slug the health checks service is exposed at. Defaults to /health.

- --HealthChecks.UIEnabled <value> NETHERMINDHEALTHCHECKSCONFIGUIENABLED

  Whether to enable the health checks UI. Allowed values: true false. Defaults to false.

- --HealthChecks.WebhooksEnabled <value>
NETHERMINDHEALTHCHECKSCONFIGWEBHOOKSENABLED

  Whether to enable web hooks. Allowed values: true false. Defaults to false.

- --HealthChecks.WebhooksPayload <value>
NETHERMINDHEALTHCHECKSCONFIGWEBHOOKSPAYLOAD

  An escaped JSON paylod to be sent to the web hook on failure.
  Defaults to:

  json
  {
    "attachments": [

```
      {
        "color": "#FFCC00",
        "pretext": "Health Check Status :warning:",
        "fields": [
          {
            "title": "Details",
            "value": "More details available at /healthchecks-ui",
            "short": false
          },
          {
            "title": "Description",
            "value": "[[DESCRIPTIONS]]",
            "short": false
          }
        ]
      }
    ]
  }
```

```
- --HealthChecks.WebhooksRestorePayload <value>
NETHERMINDHEALTHCHECKSCONFIGWEBHOOKSRESTOREPAYLOAD

  An escaped JSON paylod to be sent to the web hook on recovery.
  Defaults to:

  json
  {
    "attachments": [
      {
        "color": "#36a64f",
        "pretext": "Health Check Status :+1:",
        "fields": [
          {
            "title": "Details",
            "value": "More details available at /healthchecks-ui",
            "short": false
          },
          {
            "title": "description",
            "value": "The HealthCheck [[LIVENESS]] is recovered. Everything is
up and running.",
            "short": false
          }
        ]
      }
    ]
  }
```

```
- --HealthChecks.WebhooksUri <value> NETHERMINDHEALTHCHECKSCONFIGWEBHOOKSURI

  The web hook URL. Defaults to null.

</p>
</details>

<details>
<summary className="nd-details-heading">

Hive
```

</summary>

- --Hive.BlocksDir <value> NETHERMINDHIVECONFIGBLOCKSDIR

  The path to the directory with additional blocks. Defaults to /blocks.

- --Hive.ChainFile <value> NETHERMINDHIVECONFIGCHAINFILE

  The path to the test chain spec file. Defaults to /chain.rlp.

- --Hive.Enabled <value> NETHERMINDHIVECONFIGENABLED

  Whether to enable Hive for debugging. Allowed values: true false. Defaults to false.

- --Hive.GenesisFilePath <value> NETHERMINDHIVECONFIGGENESISFILEPATH

  The path to the genesis block file. Defaults to /genesis.json.

- --Hive.KeysDir <value> NETHERMINDHIVECONFIGKEYSDIR

  The path to the keystore directory. Defaults to /keys.

</details>

<details>
<summary className="nd-details-heading">

Init

</summary>

- --Init.AutoDump <value> NETHERMINDINITCONFIGAUTODUMP

  Auto-dump on bad blocks for diagnostics. Default combines Receipts and Rlp.

  Allowed values:

    - None
    - Receipts
    - Parity
    - Geth
    - Rlp
    - RlpLog
    - Default
    - All

  Defaults to Default.

- --Init.BadBlocksStored <value> NETHERMINDINITCONFIGBADBLOCKSSTORED

  The maximum number of bad blocks observed on the network that will be stored on disk. Defaults to 100.

- --Init.BaseDbPath <value> NETHERMINDINITCONFIGBASEDBPATH

  The base path for all Nethermind databases. Defaults to db.

- --Init.ChainSpecPath <value> NETHERMINDINITCONFIGCHAINSPECPATH

  The path to the chain spec file. Defaults to chainspec/foundation.json.

- --Init.DiagnosticMode <value> NETHERMINDINITCONFIGDIAGNOSTICMODE

  The diagnostic mode.

  Allowed values:

    - None
    - MemDb
    - RpcDb
    - ReadOnlyDb
    - VerifyRewards
    - VerifySupply
    - VerifyTrie

  Defaults to None.

- --Init.DiscoveryEnabled <value> NETHERMINDINITCONFIGDISCOVERYENABLED

  Whether to enable the node discovery. If disabled, Nethermind doesn't look for other nodes beyond the bootnodes specified. Allowed values: true false. Defaults to true.

- --Init.EnableUnsecuredDevWallet <value> NETHERMINDINITCONFIGENABLEUNSECUREDDEVWALLET

  Whether to enable the in-app wallet/keystore. Allowed values: true false. Defaults to false.

- --Init.GenesisHash <value> NETHERMINDINITCONFIGGENESISHASH

  The hash of the genesis block. If not specified, the genesis block validity is not checked which is useful in the case of ad hoc test/private networks. Defaults to null.

- --Init.HiveChainSpecPath <value> NETHERMINDINITCONFIGHIVECHAINSPECPATH

  The path to the chain spec file for Hive tests. Defaults to chainspec/test.json.

- --Init.IsMining <value> NETHERMINDINITCONFIGISMINING

  Whether to seal/mine new blocks. Allowed values: true false. Defaults to false.

- --Init.KeepDevWalletInMemory <value> NETHERMINDINITCONFIGKEEPDEVWALLETINMEMORY

  Whether to create session-only accounts and delete them on shutdown. Allowed values: true false. Defaults to false.

- --Init.KzgSetupPath <value> NETHERMINDINITCONFIGKZGSETUPPATH

  The path to KZG trusted setup file. Defaults to null.

- --Init.LogDirectory <value> NETHERMINDINITCONFIGLOGDIRECTORY

  The path to the Nethermind logs directory. Defaults to logs.

- --Init.LogFileName <value> NETHERMINDINITCONFIGLOGFILENAME

  The name of the log file. Defaults to log.txt.

- --Init.LogRules <value> NETHERMINDINITCONFIGLOGRULES

  The logs format as LogPath:LogLevel; Defaults to null.

- --Init.MemoryHint <value> NETHERMINDINITCONFIGMEMORYHINT

  The hint on the max memory limit, in bytes, to configure the database and networking memory allocations. Defaults to null.

- --Init.PeerManagerEnabled <value> NETHERMINDINITCONFIGPEERMANAGERENABLED

  Whether to connect to newly discovered peers. Allowed values: true false. Defaults to true.

- --Init.ProcessingEnabled <value> NETHERMINDINITCONFIGPROCESSINGENABLED

  Whether to download/process new blocks. Allowed values: true false. Defaults to true.

- --Init.RpcDbUrl <value> NETHERMINDINITCONFIGRPCDBURL

  The URL of the remote node used as a database source when DiagnosticMode is set to RpcDb.

- --Init.StaticNodesPath <value> NETHERMINDINITCONFIGSTATICNODESPATH

  The path to the static nodes file. Defaults to Data/static-nodes.json.

- --Init.WebSocketsEnabled <value> NETHERMINDINITCONFIGWEBSOCKETSENABLED

  Whether to enable WebSocket service for the default JSON-RPC port on startup. Allowed values: true false. Defaults to true.

</details>

<details>
<summary className="nd-details-heading">

JsonRpc

</summary>

- --JsonRpc.AdditionalRpcUrls <value> NETHERMINDJSONRPCCONFIGADDITIONALRPCURLS

  An array of additional JSON-RPC URLs to listen at with protocol and JSON-RPC namespace list. For instance, [http://localhost:8546|http;ws|eth;web3]. Defaults to [].

- --JsonRpc.BufferResponses <value> NETHERMINDJSONRPCCONFIGBUFFERRESPONSES

  Whether to buffer responses before sending them. This allows using of Content-Length instead of Transfer-Encoding: chunked. Note that it may degrade performance on large responses. The max buffered response length is 2GB. Chunked responses can be larger. Allowed values: true false. Defaults to false.

- --JsonRpc.CallsFilterFilePath <value> NETHERMINDJSONRPCCONFIGCALLSFILTERFILEPATH

  The path to a file with the list of new-line-separated JSON-RPC calls. If specified, only the calls from that file are allowed. Defaults to Data/jsonrpc.filter.

- --JsonRpc.Enabled <value> NETHERMINDJSONRPCCONFIGENABLED

  Whether to enable the JSON-RPC service. Allowed values: true false. Defaults to false.

- --JsonRpc.EnabledModules <value> NETHERMINDJSONRPCCONFIGENABLEDMODULES

  An array of JSON-RPC namespaces to enable. For instance, [debug,eth].

  Built-in namespaces:

  - admin
  - client
  - debug
  - engine
  - eth
  - evm
  - health
  - net
  - parity
  - personal
  - proof
  - rpc
  - subscribe
  - trace
  - txpool
  - web3

  Defaults to [Eth,Subscribe,Trace,TxPool,Web3,Personal,Proof,Net,Parity,Health,Rpc].

- --JsonRpc.EngineEnabledModules <value> NETHERMINDJSONRPCCONFIGENGINEENABLEDMODULES

  An array of additional JSON-RPC URLs to listen at with protocol and JSON-RPC namespace list for Engine API. Defaults to [Net,Eth,Subscribe,Web3].

- --JsonRpc.EngineHost <value> NETHERMINDJSONRPCCONFIGENGINEHOST

  The Engine API host. Defaults to 127.0.0.1.

- --JsonRpc.EnginePort <value> NETHERMINDJSONRPCCONFIGENGINEPORT

  The Engine API port. Defaults to null.

- --JsonRpc.EstimateErrorMargin <value> NETHERMINDJSONRPCCONFIGESTIMATEERRORMARGIN

  The error margin used in ethestimateGas expressed in basis points. Defaults to 150.

- --JsonRpc.EthModuleConcurrentInstances <value> NETHERMINDJSONRPCCONFIGETHMODULECONCURRENTINSTANCES

  The number of concurrent instances for non-sharable calls:

  - ethcall
  - ethestimateGas
  - ethgetLogs
  - ethnewBlockFilter
  - ethnewFilter
  - ethnewPendingTransactionFilter
  - ethuninstallFilter

  This limits the load on the CPU and I/O to reasonable levels. If the limit is exceeded, HTTP 503 is returned along with the JSON-RPC error. Defaults to the

number of logical processors.

- --JsonRpc.GasCap <value> NETHERMINDJSONRPCCONFIGGASCAP

  The gas limit for ethcall and ethestimateGas. Defaults to 100000000.

- --JsonRpc.Host <value> NETHERMINDJSONRPCCONFIGHOST

  The JSON-RPC service host. Defaults to 127.0.0.1.

- --JsonRpc.IpcUnixDomainSocketPath <value>
NETHERMINDJSONRPCCONFIGIPCUNIXDOMAINSOCKETPATH

  The path to connect a UNIX domain socket over.

- --JsonRpc.JwtSecretFile <value> NETHERMINDJSONRPCCONFIGJWTSECRETFILE

  The path to the JWT secret file required for the Engine API authentication. Defaults to keystore/jwt-secret.

- --JsonRpc.MaxBatchResponseBodySize <value>
NETHERMINDJSONRPCCONFIGMAXBATCHRESPONSEBODYSIZE

  The max batch size limit for batched JSON-RPC calls. Defaults to 33554432.

- --JsonRpc.MaxBatchSize <value> NETHERMINDJSONRPCCONFIGMAXBATCHSIZE

  The max number of JSON-RPC requests in a batch. Defaults to 1024.

- --JsonRpc.MaxLoggedRequestParametersCharacters <value>
NETHERMINDJSONRPCCONFIGMAXLOGGEDREQUESTPARAMETERSCHARACTERS

  The max number of characters of a JSON-RPC request parameter printing to the log. Defaults to null.

- --JsonRpc.MaxLogsPerResponse <value> NETHERMINDJSONRPCCONFIGMAXLOGSPERRESPONSE

  The max number of logs per response. For method ethgetLogs. If 0 then no limit. Defaults to 20000.

- --JsonRpc.MaxRequestBodySize <value> NETHERMINDJSONRPCCONFIGMAXREQUESTBODYSIZE

  The max length of HTTP request body, in bytes. Defaults to 30000000.

- --JsonRpc.MaxSimulateBlocksCap <value>
NETHERMINDJSONRPCCONFIGMAXSIMULATEBLOCKSCAP

  The max blocks count limit for ethsimulate JSON-RPC calls. Defaults to 256.

- --JsonRpc.MethodsLoggingFiltering <value>
NETHERMINDJSONRPCCONFIGMETHODSLOGGINGFILTERING

  An array of the method names not to log. Defaults to [enginenewPayloadV1,enginenewPayloadV2,enginenewPayloadV3,engineforkchoiceUpdatedV1,engineforkchoiceUpdatedV2].

- --JsonRpc.Port <value> NETHERMINDJSONRPCCONFIGPORT

  The JSON-RPC service HTTP port. Defaults to 8545.

- --JsonRpc.ReportIntervalSeconds <value>
NETHERMINDJSONRPCCONFIGREPORTINTERVALSECONDS

  The interval, in seconds, between the JSON-RPC stats report log. Defaults to

300.

- --JsonRpc.RequestQueueLimit <value> NETHERMINDJSONRPCCONFIGREQUESTQUEUELIMIT

  The max number of concurrent requests in the queue for:

  - ethcall
  - ethestimateGas
  - ethgetLogs
  - ethnewFilter
  - ethnewBlockFilter
  - ethnewPendingTransactionFilter
  - ethuninstallFilter

  0 to lift the limit. Defaults to 500.

- --JsonRpc.RpcRecorderBaseFilePath <value>
NETHERMINDJSONRPCCONFIGRPCRECORDERBASEFILEPATH

  The path to the base file for diagnostic recording. Defaults to logs/rpc.
{counter}.txt.

- --JsonRpc.RpcRecorderState <value> NETHERMINDJSONRPCCONFIGRPCRECORDERSTATE

  The diagnostic recording mode.

  Allowed values:

    - None
    - Request
    - Response
    - All

  Defaults to None.

- --JsonRpc.Timeout <value> NETHERMINDJSONRPCCONFIGTIMEOUT

  The request timeout, in milliseconds. Defaults to 20000.

- --JsonRpc.WebSocketsPort <value> NETHERMINDJSONRPCCONFIGWEBSOCKETSPORT

  The JSON-RPC service WebSockets port. Defaults to 8545.

</details>

<details>
<summary className="nd-details-heading">

KeyStore

</summary>

- --KeyStore.BlockAuthorAccount <value>
NETHERMINDKEYSTORECONFIGBLOCKAUTHORACCOUNT

  An account to use as the block author (coinbase).

- --KeyStore.Cipher <value> NETHERMINDKEYSTORECONFIGCIPHER

  See [Web3 secret storage definition][web3-secret-storage]. Defaults to aes-
128-ctr.

- --KeyStore.EnodeAccount <value> NETHERMINDKEYSTORECONFIGENODEACCOUNT

  An account to use for networking (enode). If neither this nor the EnodeKeyFile option is specified, the key is autogenerated in node.key.plain file.

- --KeyStore.EnodeKeyFile <value> NETHERMINDKEYSTORECONFIGENODEKEYFILE

  The path to the key file to use by for networking (enode). If neither this nor the EnodeAccount is specified, the key is autogenerated in node.key.plain file.

- --KeyStore.IVSize <value> NETHERMINDKEYSTORECONFIGIVSIZE

  See [Web3 secret storage definition][web3-secret-storage]. Defaults to 16.

- --KeyStore.Kdf <value> NETHERMINDKEYSTORECONFIGKDF

  See [Web3 secret storage definition][web3-secret-storage]. Defaults to scrypt.

- --KeyStore.KdfparamsDklen <value> NETHERMINDKEYSTORECONFIGKDFPARAMSDKLEN

  See [Web3 secret storage definition][web3-secret-storage]. Defaults to 32.

- --KeyStore.KdfparamsN <value> NETHERMINDKEYSTORECONFIGKDFPARAMSN

  See [Web3 secret storage definition][web3-secret-storage]. Defaults to 262144.

- --KeyStore.KdfparamsP <value> NETHERMINDKEYSTORECONFIGKDFPARAMSP

  See [Web3 secret storage definition][web3-secret-storage]. Defaults to 1.

- --KeyStore.KdfparamsR <value> NETHERMINDKEYSTORECONFIGKDFPARAMSR

  See [Web3 secret storage definition][web3-secret-storage]. Defaults to 8.

- --KeyStore.KdfparamsSaltLen <value> NETHERMINDKEYSTORECONFIGKDFPARAMSSALTLEN

  See [Web3 secret storage definition][web3-secret-storage]. Defaults to 32.

- --KeyStore.KeyStoreDirectory <value> NETHERMINDKEYSTORECONFIGKEYSTOREDIRECTORY

  The path to the keystore directory. Defaults to keystore.

- --KeyStore.KeyStoreEncoding <value> NETHERMINDKEYSTORECONFIGKEYSTOREENCODING

  See [Web3 secret storage definition][web3-secret-storage]. Defaults to UTF-8.

- --KeyStore.PasswordFiles <value> NETHERMINDKEYSTORECONFIGPASSWORDFILES

  An array of password files paths used to unlock the accounts set with UnlockAccounts. Defaults to [].

- --KeyStore.Passwords <value> NETHERMINDKEYSTORECONFIGPASSWORDS

  An array of passwords used to unlock the accounts set with UnlockAccounts. Defaults to [].

- --KeyStore.SymmetricEncrypterBlockSize <value> NETHERMINDKEYSTORECONFIGSYMMETRICENCRYPTERBLOCKSIZE

  See [Web3 secret storage definition][web3-secret-storage]. Defaults to 128.

- --KeyStore.SymmetricEncrypterKeySize <value> NETHERMINDKEYSTORECONFIGSYMMETRICENCRYPTERKEYSIZE

  See [Web3 secret storage definition][web3-secret-storage]. Defaults to 128.

- --KeyStore.TestNodeKey <value> NETHERMINDKEYSTORECONFIGTESTNODEKEY

  A plaintext private key to use for testing purposes.

- --KeyStore.UnlockAccounts <value> NETHERMINDKEYSTORECONFIGUNLOCKACCOUNTS

  An array of accounts to unlock on startup using passwords either in PasswordFiles and Passwords. Defaults to [].

</details>

<details>
<summary className="nd-details-heading">

Merge

</summary>

- --Merge.BuilderRelayUrl <value> NETHERMINDMERGECONFIGBUILDERRELAYURL

  The URL of a builder relay. If specified, blocks are sent to the relay. Defaults to null.

- --Merge.CollectionsPerDecommit <value> NETHERMINDMERGECONFIGCOLLECTIONSPERDECOMMIT

  Request the garbage collector (GC) to release the process memory.

  Allowed values:

  - -1 to disable
  - 0 to release every time
  - A positive number to release memory after that many Engine API calls

  Defaults to 25.

- --Merge.CompactMemory <value> NETHERMINDMERGECONFIGCOMPACTMEMORY

  The memory compaction mode. When set to Full, compacts the large object heap (LOH) if SweepMemory is set to Gen2.

  Allowed values:

    - No
    - Yes
    - Full

  Defaults to Yes.

- --Merge.Enabled <value> NETHERMINDMERGECONFIGENABLED

  Whether to enable the Merge hard fork. Allowed values: true false. Defaults to true.

- --Merge.FinalTotalDifficulty <value> NETHERMINDMERGECONFIGFINALTOTALDIFFICULTY

  The total difficulty of the last PoW block. Must be greater than or equal to the terminal total difficulty (TTD). Defaults to null.

- --Merge.PrioritizeBlockLatency <value>

NETHERMINDMERGECONFIGPRIORITIZEBLOCKLATENCY

  Whether to reduce block latency by disabling garbage collection during Engine API calls. Allowed values: true false. Defaults to true.

- --Merge.SweepMemory <value> NETHERMINDMERGECONFIGSWEEPMEMORY

  The garbage collection (GC) mode between Engine API calls.

  Allowed values:

    - NoGC
    - Gen0
    - Gen1
    - Gen2

  Defaults to Gen1.

- --Merge.TerminalBlockHash <value> NETHERMINDMERGECONFIGTERMINALBLOCKHASH

  The terminal PoW block hash used for the transition. Defaults to null.

- --Merge.TerminalBlockNumber <value> NETHERMINDMERGECONFIGTERMINALBLOCKNUMBER

  The terminal PoW block number used for the transition.

- --Merge.TerminalTotalDifficulty <value> NETHERMINDMERGECONFIGTERMINALTOTALDIFFICULTY

  The terminal total difficulty (TTD) used for the transition. Defaults to null.

</details>

<details>
<summary className="nd-details-heading">

Metrics

</summary>

- --Metrics.CountersEnabled <value> NETHERMINDMETRICSCONFIGCOUNTERSENABLED

  Whether to publish metrics using .NET diagnostics that can be collected with dotnet-counters. Allowed values: true false. Defaults to false.

- --Metrics.Enabled <value> NETHERMINDMETRICSCONFIGENABLED

  Whether to publish various metrics to Prometheus Pushgateway at a given interval. Allowed values: true false. Defaults to false.

- --Metrics.EnableDbSizeMetrics <value> NETHERMINDMETRICSCONFIGENABLEDBSIZEMETRICS

  Whether to publish database size metrics. Allowed values: true false. Defaults to true.

- --Metrics.ExposeHost <value> NETHERMINDMETRICSCONFIGEXPOSEHOST

  The IP address to expose Prometheus metrics at. The value of + means listening on all available hostnames. Setting this to localhost prevents remote access. Defaults to +.

- --Metrics.ExposePort <value> NETHERMINDMETRICSCONFIGEXPOSEPORT

  The port to expose Prometheus metrics at. Defaults to null.

- --Metrics.IntervalSeconds <value> NETHERMINDMETRICSCONFIGINTERVALSECONDS

  The frequency of pushing metrics to Prometheus, in seconds. Defaults to 5.

- --Metrics.NodeName <value> NETHERMINDMETRICSCONFIGNODENAME

  The name to display on the Grafana dashboard. Defaults to "Nethermind".

- --Metrics.PushGatewayUrl <value> NETHERMINDMETRICSCONFIGPUSHGATEWAYURL

  The Prometheus Pushgateway instance URL.

</details>

<details>
<summary className="nd-details-heading">

Mining

</summary>

- --Mining.Enabled <value> NETHERMINDMININGCONFIGENABLED

  Whether to produce blocks. Allowed values: true false. Defaults to false.

- --Mining.Signer <value> NETHERMINDMININGCONFIGSIGNER

  Url for an external signer like clef: https://github.com/ethereum/go-ethereum/blob/master/cmd/clef/tutorial.md Defaults to null.

</details>

<details>
<summary className="nd-details-heading">

Network

</summary>

- --Network.Bootnodes <value> NETHERMINDNETWORKCONFIGBOOTNODES

  A comma-separated enode list to be used as boot nodes.

- --Network.DiagTracerEnabled <value> NETHERMINDNETWORKCONFIGDIAGTRACERENABLED

  Whether to enable a verbose diagnostic tracing. Allowed values: true false. Defaults to false.

- --Network.DiscoveryDns <value> NETHERMINDNETWORKCONFIGDISCOVERYDNS

  Use tree is available through a DNS name. For the default of <chain name>.ethdisco.net, leave unspecified. Defaults to null.

- --Network.DiscoveryPort <value> NETHERMINDNETWORKCONFIGDISCOVERYPORT

  The UDP port number for incoming discovery connections. It's recommended to

keep it the same as the TCP port (P2PPort) because other values have not been tested yet. Defaults to 30303.

- --Network.EnableUPnP <value> NETHERMINDNETWORKCONFIGENABLEUPNP

  Whether to enable automatic port forwarding via UPnP. Allowed values: true false. Defaults to false.

- --Network.ExternalIp <value> NETHERMINDNETWORKCONFIGEXTERNALIP

  The external IP. Use only when the external IP cannot be resolved automatically. Defaults to null.

- --Network.LocalIp <value> NETHERMINDNETWORKCONFIGLOCALIP

  The local IP. Use only when the local IP cannot be resolved automatically. Defaults to null.

- --Network.MaxActivePeers <value> NETHERMINDNETWORKCONFIGMAXACTIVEPEERS

  The max allowed number of connected peers. Defaults to 50.

- --Network.MaxNettyArenaCount <value> NETHERMINDNETWORKCONFIGMAXNETTYARENACOUNT

  The maximum DotNetty arena count. Increasing this on a high-core CPU without increasing the memory budget may reduce chunk size so much that it causes a huge memory allocation. Defaults to 8.

- --Network.NettyArenaOrder <value> NETHERMINDNETWORKCONFIGNETTYARENAORDER

  The size of the DotNetty arena order. -1 to depend on the memory hint. Defaults to -1.

- --Network.OnlyStaticPeers <value> NETHERMINDNETWORKCONFIGONLYSTATICPEERS

  Whether to use static peers only. Allowed values: true false. Defaults to false.

- --Network.P2PPort <value> NETHERMINDNETWORKCONFIGP2PPORT

  The TCP port for incoming P2P connections. Defaults to 30303.

- --Network.PriorityPeersMaxCount <value> NETHERMINDNETWORKCONFIGPRIORITYPEERSMAXCOUNT

  The max number of priority peers. Can be overridden by a plugin. Defaults to 0.

- --Network.StaticPeers <value> NETHERMINDNETWORKCONFIGSTATICPEERS

  A list of peers to keep connection for. Static peers are affected by MaxActivePeers. Defaults to null.

</details>

<details>
<summary className="nd-details-heading">

Optimism

</summary>

- --Optimism.SequencerUrl <value> NETHERMINDOPTIMISMCONFIGSEQUENCERURL

  Sequencer address Defaults to null.

</details>

<details>
<summary className="nd-details-heading">

Pruning

</summary>

- --Pruning.AvailableSpaceCheckEnabled <value>
NETHERMINDPRUNINGCONFIGAVAILABLESPACECHECKENABLED

  Whether to enables available disk space check. Allowed values: true false.
Defaults to true.

- --Pruning.CacheMb <value> NETHERMINDPRUNINGCONFIGCACHEMB

  The in-memory cache size, in MB. The bigger the cache size, the bigger the
disk space savings. Defaults to 1024.

- --Pruning.FullPruningCompletionBehavior <value>
NETHERMINDPRUNINGCONFIGFULLPRUNINGCOMPLETIONBEHAVIOR

  The behavior after pruning completion:

  - None: Do nothing.
  - ShutdownOnSuccess: Shut Nethermind down if pruning has succeeded but leave
it running if failed.
  - AlwaysShutdown: Shut Nethermind down when pruning completes, regardless of
its status.

  Allowed values:

    - None
    - ShutdownOnSuccess
    - AlwaysShutdown

  Defaults to None.

- --Pruning.FullPruningDisableLowPriorityWrites <value>
NETHERMINDPRUNINGCONFIGFULLPRUNINGDISABLELOWPRIORITYWRITES

  Whether to disable low-priority for pruning writes. Full pruning uses low-
priority write operations to prevent blocking block processing. If block
processing is not high-priority, set this option to true for faster pruning.
Allowed values: true false. Defaults to false.

- --Pruning.FullPruningMaxDegreeOfParallelism <value>
NETHERMINDPRUNINGCONFIGFULLPRUNINGMAXDEGREEOFPARALLELISM

  The max number of parallel tasks that can be used by full pruning:

  Allowed values:

  - -1 to use the number of logical processors
  - 0 to use 25% of logical processors
  - 1 to run on single thread

The recommended value depends on the type of the node:

- If the node needs to be responsive (serves for RPC or validator), then the recommended value is 0 or -1.
- If the node doesn't have many other responsibilities but needs to be able to follow the chain reliably without any delays and produce live logs, the 0 or 1 is recommended.
- If the node doesn't have to be responsive, has very fast I/O (like NVMe) and the shortest pruning time is to be achieved, then -1 is recommended. Defaults to 0.

- --Pruning.FullPruningMemoryBudgetMb <value> NETHERMINDPRUNINGCONFIGFULLPRUNINGMEMORYBUDGETMB

The memory budget, in MB, used for the trie visit. Increasing this value significantly reduces the IOPS requirement at the expense of memory usage. 0 to disable. Defaults to 4000.

- --Pruning.FullPruningMinimumDelayHours <value> NETHERMINDPRUNINGCONFIGFULLPRUNINGMINIMUMDELAYHOURS

The minimum delay, in hours, between full pruning operations not to exhaust disk writes. Defaults to 240.

- --Pruning.FullPruningThresholdMb <value> NETHERMINDPRUNINGCONFIGFULLPRUNINGTHRESHOLDMB

The threshold, in MB, to trigger full pruning. Depends on Mode and FullPruningTrigger. Defaults to 256000.

- --Pruning.FullPruningTrigger <value> NETHERMINDPRUNINGCONFIGFULLPRUNINGTRIGGER

The full pruning trigger:

- Manual: Triggered manually.
- StateDbSize: Trigger when the state DB size is above the threshold.
- VolumeFreeSpace: Trigger when the free disk space where the state DB is stored is below the threshold.

Allowed values:

- Manual
- StateDbSize
- VolumeFreeSpace

Defaults to Manual.

- --Pruning.Mode <value> NETHERMINDPRUNINGCONFIGMODE

The pruning mode:

- None: No pruning (full archive)
- Memory: In-memory pruning
- Full: Full pruning
- Hybrid: Combined in-memory and full pruning

Allowed values:

- None
- Memory
- Full
- Hybrid

Defaults to Hybrid.

- --Pruning.PersistenceInterval <value> NETHERMINDPRUNINGCONFIGPERSISTENCEINTERVAL

  The block persistence frequency. If set to N, it caches after each Nth block even if not required by cache memory usage. Defaults to 8192.

- --Pruning.PruningBoundary <value> NETHERMINDPRUNINGCONFIGPRUNINGBOUNDARY

  Past N state before state gets pruned Used to determine how old of a state to keep from the head. Defaults to 64.

- --Pruning.TrackedPastKeyCountMemoryRatio <value> NETHERMINDPRUNINGCONFIGTRACKEDPASTKEYCOUNTMEMORYRATIO

  [TECHNICAL] Ratio of memory out of CacheMb to allocate for LRU used to track past keys for live pruning. Defaults to 0.1.

</details>

<details>
<summary className="nd-details-heading">

Receipt

</summary>

- --Receipt.CompactReceiptStore <value> NETHERMINDRECEIPTCONFIGCOMPACTRECEIPTSTORE

  Whether to compact receipts database size at the expense of RPC performance. Allowed values: true false. Defaults to true.

- --Receipt.CompactTxIndex <value> NETHERMINDRECEIPTCONFIGCOMPACTTXINDEX

  Whether to compact receipts transaction index database size at the expense of RPC performance. Allowed values: true false. Defaults to true.

- --Receipt.ReceiptsMigration <value> NETHERMINDRECEIPTCONFIGRECEIPTSMIGRATION

  Whether to migrate the receipts database to the new schema. Allowed values: true false. Defaults to false.

- --Receipt.StoreReceipts <value> NETHERMINDRECEIPTCONFIGSTORERECEIPTS

  Whether to store receipts after a new block is processed. This setting is independent from downloading receipts in fast sync mode. Allowed values: true false. Defaults to true.

- --Receipt.TxLookupLimit <value> NETHERMINDRECEIPTCONFIGTXLOOKUPLIMIT

  The number of recent blocks to maintain transaction index for. 0 to never remove indices, -1 to never index. Defaults to 2350000.

</details>

<details>
<summary className="nd-details-heading">

Seq

</summary>

- --Seq.ApiKey <value> NETHERMINDSEQCONFIGAPIKEY

  The Seq API key.

- --Seq.MinLevel <value> NETHERMINDSEQCONFIGMINLEVEL

  The min log level to sent to Seq. Defaults to Off.

- --Seq.ServerUrl <value> NETHERMINDSEQCONFIGSERVERURL

  The Seq instance URL. Defaults to http://localhost:5341.

</details>

<details>
<summary className="nd-details-heading">

Snapshot

</summary>

- --Snapshot.Checksum <value> NETHERMINDSNAPSHOTCONFIGCHECKSUM

  The SHA-256 checksum of the snapshot file. Defaults to null.

- --Snapshot.DownloadUrl <value> NETHERMINDSNAPSHOTCONFIGDOWNLOADURL

  The URL of the snapshot file. Defaults to null.

- --Snapshot.Enabled <value> NETHERMINDSNAPSHOTCONFIGENABLED

  Whether to enable the Snapshot plugin. Allowed values: true false. Defaults to false.

- --Snapshot.SnapshotDirectory <value> NETHERMINDSNAPSHOTCONFIGSNAPSHOTDIRECTORY

  The path to the directory to store the snapshot file. Defaults to snapshot.

- --Snapshot.SnapshotFileName <value> NETHERMINDSNAPSHOTCONFIGSNAPSHOTFILENAME

  The name of the snapshot file. Defaults to snapshot.zip.

</details>

<details>
<summary className="nd-details-heading">

Sync

</summary>

- --Sync.AncientBodiesBarrier <value> NETHERMINDSYNCCONFIGANCIENTBODIESBARRIER

  The earliest body downloaded with fast sync when DownloadBodiesInFastSync is set to true. The actual value is determined as follows:

max{ 1, min{ PivotNumber, AncientBodiesBarrier } }

Defaults to 0.

- --Sync.AncientReceiptsBarrier <value> NETHERMINDSYNCCONFIGANCIENTRECEIPTSBARRIER

The earliest receipt downloaded with fast sync when DownloadReceiptsInFastSync is set to true. The actual value is determined as follows:

max{ 1, min{ PivotNumber, max{ AncientBodiesBarrier, AncientReceiptsBarrier } } }

Defaults to 0.

- --Sync.BlocksDbTuneDbMode <value> NETHERMINDSYNCCONFIGBLOCKSDBTUNEDBMODE

Configure the blocks database for write optimizations during sync.

Allowed values:

- Default
- WriteBias
- HeavyWrite
- AggressiveHeavyWrite
- DisableCompaction
- EnableBlobFiles
- HashDb

Defaults to EnableBlobFiles.

- --Sync.DownloadBodiesInFastSync <value> NETHERMINDSYNCCONFIGDOWNLOADBODIESINFASTSYNC

Whether to download the block bodies in the Fast sync mode. Allowed values: true false. Defaults to true.

- --Sync.DownloadHeadersInFastSync <value> NETHERMINDSYNCCONFIGDOWNLOADHEADERSINFASTSYNC

Whether to download the old block headers in the Fast sync mode. If false, Nethermind downloads only recent blocks headers. Allowed values: true false. Defaults to true.

- --Sync.DownloadReceiptsInFastSync <value> NETHERMINDSYNCCONFIGDOWNLOADRECEIPTSINFASTSYNC

Whether to download receipts in the Fast sync mode. This slows down the process by a few hours but allows to interact with dApps that perform extensive historical logs searches. Allowed values: true false. Defaults to true.

- --Sync.ExitOnSynced <value> NETHERMINDSYNCCONFIGEXITONSYNCED

Whether to shut down Nethermind once sync is finished. Allowed values: true false. Defaults to false.

- --Sync.ExitOnSyncedWaitTimeSec <value> NETHERMINDSYNCCONFIGEXITONSYNCEDWAITTIMESEC

The time, in seconds, to wait before shutting down Nethermind once sync is finished. Defaults to 60.

- --Sync.FastSync <value> NETHERMINDSYNCCONFIGFASTSYNC

Whether to use the Fast sync mode (the eth/63 synchronization algorithm). Allowed values: true false. Defaults to false.

- --Sync.FastSyncCatchUpHeightDelta <value> NETHERMINDSYNCCONFIGFASTSYNCCATCHUPHEIGHTDELTA

In Fast sync mode, the min height threshold limit up to which the Full sync, if already on, stays on when the chain is behind the network head. If the limit is exceeded, it switches back to Fast sync. For regular usage scenarios, setting this value lower than 32 is not recommended as this can cause issues with chain reorgs. Note that the last 2 blocks are always processed in Full sync, so setting it lower than 2 has no effect. Defaults to 8192.

- --Sync.FixReceipts <value> NETHERMINDSYNCCONFIGFIXRECEIPTS

Whether to enable receipts validation that checks for receipts that might be missing because of a bug. If needed, receipts are downloaded from the network. If true, the pivot number must be same one used originally as it's used as a cut-off point. Allowed values: true false. Defaults to false.

- --Sync.FixTotalDifficulty <value> NETHERMINDSYNCCONFIGFIXTOTALDIFFICULTY

Whether to recalculate the total difficulty from FixTotalDifficultyStartingBlock to FixTotalDifficultyLastBlock. Allowed values: true false. Defaults to false.

- --Sync.FixTotalDifficultyLastBlock <value> NETHERMINDSYNCCONFIGFIXTOTALDIFFICULTYLASTBLOCK

The last block to recalculate the total difficulty for. If not specified, the best known block is used.
Defaults to null.

- --Sync.FixTotalDifficultyStartingBlock <value> NETHERMINDSYNCCONFIGFIXTOTALDIFFICULTYSTARTINGBLOCK

The first block to recalculate the total difficulty for. Defaults to 1.

- --Sync.MaxAttemptsToUpdatePivot <value> NETHERMINDSYNCCONFIGMAXATTEMPTSTOUPDATEPIVOT

The max number of attempts to update the pivot block based on the FCU message from the consensus client. Defaults to 2147483647.

- --Sync.MaxProcessingThreads <value> NETHERMINDSYNCCONFIGMAXPROCESSINGTHREADS

The max number of threads used for syncing. 0 to use the number of logical processors. Defaults to 0.

- --Sync.NetworkingEnabled <value> NETHERMINDSYNCCONFIGNETWORKINGENABLED

Whether to connect to peers and sync. Allowed values: true false. Defaults to true.

- --Sync.NonValidatorNode <value> NETHERMINDSYNCCONFIGNONVALIDATORNODE

Whether to operate as a non-validator. If true, the DownloadReceiptsInFastSync and DownloadBodiesInFastSync can be set to false. Allowed values: true false. Defaults to false.

- --Sync.PivotHash <value> NETHERMINDSYNCCONFIGPIVOTHASH

The hash of the pivot block for the Fast sync mode. Defaults to null.

- --Sync.PivotNumber <value> NETHERMINDSYNCCONFIGPIVOTNUMBER

  The number of the pivot block for the Fast sync mode. Defaults to 0.

- --Sync.PivotTotalDifficulty <value> NETHERMINDSYNCCONFIGPIVOTTOTALDIFFICULTY

  The total difficulty of the pivot block for the Fast sync mode. Defaults to null.

- --Sync.SnapSync <value> NETHERMINDSYNCCONFIGSNAPSYNC

  Whether to use the Snap sync mode. Allowed values: true false. Defaults to false.

- --Sync.SnapSyncAccountRangePartitionCount <value> NETHERMINDSYNCCONFIGSNAPSYNCACCOUNTRANGEPARTITIONCOUNT

  The number of account range partitions to create. Increases the Snap sync request concurrency. Allowed values are between between 1 and 256. Defaults to 8.

- --Sync.StrictMode <value> NETHERMINDSYNCCONFIGSTRICTMODE

  Whether to disable some optimizations and do a more extensive sync. Useful when sync state is corrupted. Allowed values: true false. Defaults to false.

- --Sync.SynchronizationEnabled <value> NETHERMINDSYNCCONFIGSYNCHRONIZATIONENABLED

  Whether to download and process new blocks. Allowed values: true false. Defaults to true.

- --Sync.TuneDbMode <value> NETHERMINDSYNCCONFIGTUNEDBMODE

  Configure the database for write optimizations during sync. Significantly reduces the total number of writes and sync time if you are not network limited.

  Allowed values:

    - Default
    - WriteBias
    - HeavyWrite
    - AggressiveHeavyWrite
    - DisableCompaction
    - EnableBlobFiles
    - HashDb

  Defaults to HeavyWrite.

- --Sync.UseGethLimitsInFastBlocks <value> NETHERMINDSYNCCONFIGUSEGETHLIMITSINFASTBLOCKS

  Whether to make smaller requests, in Fast Blocks mode, to avoid Geth from disconnecting. On the Geth-heavy networks (e.g., Mainnet), it's a desired behavior while on Nethermind- or OpenEthereum-heavy networks (Aura), it slows down the sync by a factor of 4. Allowed values: true false. Defaults to true.

</p>
</details>

<details>
<summary className="nd-details-heading">

TraceStore

</summary>

- --TraceStore.BlocksToKeep <value> NETHERMINDTRACESTORECONFIGBLOCKSTOKEEP

  The number of blocks to store, counting from the head. If 0, all traces of the processed blocks are stored. Defaults to 10000.

- --TraceStore.DeserializationParallelization <value> NETHERMINDTRACESTORECONFIGDESERIALIZATIONPARALLELIZATION

  The max parallelization when deserialization requests the tracefilter method. 0 to use the number of logical processors. If you experience a resource shortage, set to a low number. Defaults to 0.

- --TraceStore.Enabled <value> NETHERMINDTRACESTORECONFIGENABLED

  Whether to enable the TraceStore plugin. If enabled, traces come from the database whenever possible. Allowed values: true false. Defaults to false.

- --TraceStore.TraceTypes <value> NETHERMINDTRACESTORECONFIGTRACETYPES

  The type of traces to store.

  Allowed values:

    - None
    - VmTrace
    - StateDiff
    - Trace
    - Rewards
    - All

  Defaults to Trace, Rewards.

</details>

<details>
<summary className="nd-details-heading">

TxPool

</summary>

- --TxPool.BlobCacheSize <value> NETHERMINDTXPOOLCONFIGBLOBCACHESIZE

  The max number of full blob transactions cached in memory. The default value uses max 200MB for 6 blobs where one blob is 33MB (256 128KB) Defaults to 256.

- --TxPool.BlobsSupport <value> NETHERMINDTXPOOLCONFIGBLOBSSUPPORT

  Blobs support mode:

  - Disabled: No support for blob transactions
  - InMemory: Blob transactions stored only in memory
  - Storage: Blob transactions stored in db
  - StorageWithReorgs: Blob transactions stored in db with support for restoring reorganized blob transactions to blob pool

  Allowed values:

    - Disabled
    - InMemory
    - Storage
    - StorageWithReorgs

  Defaults to StorageWithReorgs.

- --TxPool.GasLimit <value> NETHERMINDTXPOOLCONFIGGASLIMIT

  The max transaction gas allowed. Defaults to null.

- --TxPool.HashCacheSize <value> NETHERMINDTXPOOLCONFIGHASHCACHESIZE

  The max number of cached hashes of already known transactions. Set automatically by the memory hint. Defaults to 524288.

- --TxPool.InMemoryBlobPoolSize <value> NETHERMINDTXPOOLCONFIGINMEMORYBLOBPOOLSIZE

  The max number of full blob transactions stored in memory. Used only if persistent storage is disabled. Defaults to 512.

- --TxPool.MaxPendingBlobTxsPerSender <value> NETHERMINDTXPOOLCONFIGMAXPENDINGBLOBTXSPERSENDER

  The max number of pending blob transactions per single sender. 0 to lift the limit. Defaults to 16.

- --TxPool.MaxPendingTxsPerSender <value> NETHERMINDTXPOOLCONFIGMAXPENDINGTXSPERSENDER

  The max number of pending transactions per single sender. 0 to lift the limit. Defaults to 0.

- --TxPool.MinBaseFeeThreshold <value> NETHERMINDTXPOOLCONFIGMINBASEFEETHRESHOLD

  The minimal percentage of the current base fee that must be surpassed by the max fee (maxfeepergas) for the transaction to be broadcasted. Defaults to 70.

- --TxPool.PeerNotificationThreshold <value> NETHERMINDTXPOOLCONFIGPEERNOTIFICATIONTHRESHOLD

  The average percentage of transaction hashes from persistent broadcast sent to a peer together with hashes of the last added transactions. Defaults to 5.

- --TxPool.PersistentBlobStorageSize <value> NETHERMINDTXPOOLCONFIGPERSISTENTBLOBSTORAGESIZE

  The max number of full blob transactions stored in the database (increasing the number of transactions in the blob pool also results in higher memory usage). The default value uses max 13GB for 6 blobs where one blob is 2GB (16386 128KB). Defaults to 16384.

- --TxPool.ReportMinutes <value> NETHERMINDTXPOOLCONFIGREPORTMINUTES

  The current transaction pool state reporting interval, in minutes. Defaults to null.

- --TxPool.Size <value> NETHERMINDTXPOOLCONFIGSIZE

  The max number of transactions held in the mempool (the more transactions in the mempool, the more memory used). Defaults to 2048.

</details>

<details>
<summary className="nd-details-heading">

Wallet

</summary>

- --Wallet.DevAccounts <value> NETHERMINDWALLETCONFIGDEVACCOUNTS

  The number of autogenerated developer accounts to work with. Developer accounts have private keys from 00...01 to 00...n. Defaults to 10.

</details>

<!--[end autogen]-->

Environment variables

To configure Nethermind using environment variables, the following naming convention is used in all uppercase:

text
NETHERMIND{NAMESPACE}CONFIG{PROPERTYNAME}

For instance, the environment variable equivalent of the command line --JsonRpc.JwtSecretFile option is NETHERMINDJSONRPCCONFIGJWTSECRETFILE. For the list of configuration namespaces and their options, see Options by namespaces.

Configuration file

The configuration file is a JSON file with .cfg extension. The bundled configuration files are located in the configs directory and named after the network they are used for. For instance, see the Mainnet configuration file mainnet.cfg.

[web3-secret-storage]: https://ethereum.org/en/developers/docs/data-structures-and-encoding/web3-secret-storage

database.md:

---
title: Database
sidebarposition: 5
---

import Tabs from "@theme/Tabs";
import TabItem from "@theme/TabItem";

Nethermind uses the RocksDB database to store the state. By default, the database is located in the
same directory where the Nethermind executable is. You can change the database location using the -d, --baseDbPath command line option.

Database directory structure

Directory	Description
blockInfos	Information about blocks at each level of the block tree

(canonical chain and branches) |
blocks	Block bodies (block transactions and uncles)
bloom	Bloom indices for fast log searches
canonicalHashTrie	LES protocol related data
code	Contract bytecodes
discoveryNodes	Peers discovered via discovery protocol - used for quick peering after restarts (you can copy this DB between nodes to speed up peering)
headers	Block headers only
pendingTx	The second level cache of pending transactions/mempool (the first level is in memory). Wiped out on each restart.
peers	Additional sync peers information (like peer reputation) - you can copy this DB between nodes to speed up peering on fresh sync
receipts	Transaction receipts
state	Blockchain state including accounts and contract storage (Patricia trie nodes)

You can use rsync between your nodes to clone the database (One of our users copied the entire 4.5TB archive state this
way while the node was running and only stopped the node for the very last stage of rsync). You can also copy
the database between Linux, Windows, and macOS.

Database size

Below is a comprehensive list of the supported chains, along with a detailed breakdown of their respective database directories. For reference, the database sizes listed have been determined using the standard configurations provided.

<!--[start autogen]-->

<Tabs>
<TabItem value="mainnet" label="Mainnet">

- state: 157 GB
- receipts: 206 GB
- blocks: 589 GB
- bloom: 6.4 GB
- headers: 8.8 GB
- code: 4.6 GB
- blobTransactions: 1.5 GB
- ...
- Total: 974 GB

</TabItem>
<TabItem value="sepolia" label="Sepolia">

- state: 41 GB
- receipts: 38 GB
- blocks: 267 GB
- bloom: 2.1 GB
- headers: 2.3 GB
- code: 6.2 GB
- blobTransactions: 639 MB
- ...
- Total: 356 GB

</TabItem>
<TabItem value="holesky" label="Holesky">

- state: 18 GB
- receipts: 13 GB
- blocks: 52 GB
- bloom: 689 MB

- headers: 892 MB
- code: 459 MB
- blobTransactions: 311 MB
- ...
- Total: 84 GB

</TabItem>
<TabItem value="gnosis" label="Gnosis">

- state: 65 GB
- receipts: 216 GB
- blocks: 197 GB
- bloom: 9.1 GB
- headers: 11 GB
- code: 663 MB
- blobTransactions: 75 MB
- ...
- Total: 500 GB

</TabItem>
<TabItem value="chiado" label="Chiado">

- state: 2.5 GB
- receipts: 1.4 GB
- blocks: 8.7 GB
- bloom: 3.0 GB
- headers: 2.2 GB
- code: 63 MB
- blobTransactions: 246 MB
- ...
- Total: 19 GB

</TabItem>
<TabItem value="energyweb" label="Energyweb">

- state: 27 GB
- receipts: 4.4 GB
- blocks: 24 GB
- bloom: 9.7 GB
- headers: 6.9 GB
- code: 14 MB
- blobTransactions:
- ...
- Total: 74 GB

</TabItem>
<TabItem value="volta" label="Volta">

- state: 34 GB
- receipts: 8.3 GB
- blocks: 33 GB
- bloom: 9.0 GB
- headers: 6.8 GB
- code: 94 MB
- blobTransactions:
- ...
- Total: 93 GB

</TabItem>
</Tabs>

<!--[end autogen]-->

Reducing database size

The Nethermind database can experience substantial growth over time, starting from an initial size of approximately 650
GB. As a result, many node setups are configured to run on 1 TB disks. However, even with settings
designed to slow the growth rate, these disks may eventually run out of free space.

The current options to reduce the database size are as follows:

- Resyncing database from scratch
- Pruning

The table below presents a short comparison of these methods including possible fine-tuning of each method. Data was
fetched from a node running on a machine with the below specifications:

- Node.js: v1.18.0
- Consensus client: Lighthouse
- CPU: AMD EPYC 7713 (16 cores allocated for the VM)
- RAM: 64 GB
- Disk size: 1.2 TB
- Disk IOPS: 70,000 to 80,000

Metric	Resync	Pruning	Pruning and memory budget (4 GB)
Execution time	4h	24h	12h
Minimum free disk space	N/A. You can execute resync even if there is 0 free space (avoid such a case).	250 GB	250 GB
Attestation rate drop	100%. No attestation rewards during that time or highly reduced.	5â “10% during that time	N/A
Average block processing time of new blocks during the process	N/A. New blocks are processed after state but are significantly slower until old bodies/receipts are downloaded. Afterward, average about 0.35s.	0.7s	1.0s
Is the node online during the process?	No, unless the state is synced.	Yes. The node follows the chain, and all modules are still enabled.	Yes. The node follows chain and all modules are still enabled.

The command used for testing disk IOPS was as follows:

bash
fio --randrepeat=1 --ioengine=libaio --direct=1 --gtodreduce=1 --name=test --filename=test --bs=4k --iodepth=64 --size=4G --readwrite=randrw

logs.md:

---
title: Logs
sidebarposition: 4

:::warning
This article requires a revision.
:::

Log config file location

Logging in Nethermind is done via NLog library that can be configured by editing the NLog.config file.

| Environment Type
| NLog.config location |

|---
----|---|
| built from src - Debug mode
| src\Nethermind\Nethermind.Runner\bin\Debug\netcoreapp3.1\NLog.config |
| built from src - Release mode
| src\Nethermind\Nethermind.Runner\bin\Release\netcoreapp3.1\NLog.config |
| PPA
| /usr/share/nethermind/NLog.config |
| Docker
| /nethermind/NLog.config |
| from downloads page | top level directory after
unzipping the package |
| from GitHub releases page | top level directory after unzipping the package
|
| dAppNode
| ? \[to be documented] |

Log config file syntax

Detailed NLog configuration options can be found
here: https://nlog-project.org/config/

Config or CLI log rules

Simple logging rules can be added through configuration file or command line
argument.

For example this would add Trace level logs to any logger under Synchronization
module and Debug level logs
for BlockTree from Blockchain module:\
--Init.LogRules Synchronization.:Trace;Blockchain.BlockTree:Debug

Global logging override

Additionally there are global logging override that you can use temporarily:

Command line override	Log level
./Nethermind.Runner --config mainnet --log TRACE	TRACE
./Nethermind.Runner --config mainnet --log DEBUG	DEBUG
./Nethermind.Runner --config mainnet --log INFO	INFO
./Nethermind.Runner --config mainnet --log WARN	WARN
./Nethermind.Runner --config mainnet --log ERROR	ERROR

JSON RPC logging level

This can be done by including these lines in the logging configuration file:

<logger name="JsonRpc." minlevel="Error" writeTo="file-async"/>
<logger name="JsonRpc." minlevel="Error" writeTo="auto-colored-console-async"
final="true"/>
<logger name="JsonRpc." final="true"/>

Enterprise Logging

See how to configure Seq here

Explaining Nethermind logs

You can check the supported operating systems, architectures and hardware
requirements
here: system-requirements.md

After the node starts, you will see some initial info about the node and then the sync will start. GÃ¶rli fast sync uses
a fast blocks sync mode initially. The fast blocks sync picks some known pivot block from the past and
downloads headers, bodies, and receipts downwards all the way to genesis block. All blocks from 0 to
the pivot block are showed as Old Headers in the fast blocks sync logs. The console display shows the number
growing from 0 to pivot, but this is just to make the display more user-friendly.

You will see some information about the sync progress, like below:

1. Shows the number of already downloaded headers, bodies and receipts under the name Downloaded out of all to
   be downloaded in the fast blocks stage.
2. Shows the current queue of already downloaded blocks, headers and receipts waiting for being saved to the
   database.
3. Shows the current download speed (blocks per second - bps).
4. Shows the average download speed (blocks per second - bps).

!Fast blocks sync logs

When the fast blocks stage finishes, there will be some period of downloading blocks between the pivot and
thelatest blocks which will have some additional info:

1. Shows the last entry from the fast blocks stage.
2. Shows the mode transition moment.
3. Displays the speed (in blocks per second) of all headers, bodies and receipts at the same time.
4. Additional info will appear every 30000 blocks with information about the GÃ¶rli epoch being stored.

!GÃ¶rli fast sync logs

After the fast sync part finished, the node will transition to the state sync stage when the state trie is being
downloaded. Much information is displayed about the progress, as this process may take a long time on mainnet (a few
hours).

1. Total elapsed time in state sync is displayed.
2. The total percentage of downloaded DB size is displayed (on mainnet the sync finishes around 34GB in March 2020, on
   GÃ¶rli around 800MB).
3. branches stands for the percentage of downloaded branches.
4. Download speed in kilobytes per second is displayed.
5. accounts stands for the number of accounts data downloaded already.
6. nodes stands for the number of Patricia trie nodes downloaded by the sync process.
7. diagnostic shows the time spent in the DB write / read access. The higher the value, the worse. It may get much
   worse if you restart the node during the sync process, as we need to recreate some caches then by reading data from
   the DB.

!GÃ¶rli state sync logs

When the state sync is nearing completion, you may see a series of branch sync information reloading many times from
0% to nearly 100%. This is the node trying to retrieve the few remaining state

nodes and progressing with the head block
rapidly:

!GÃ¶rli branch sync logs

At some point, the entire state is downloaded and the node enters the full sync mode and will allow you to issue CLI /
Web3 queries and send / receive transactionsðŸ¥³

1. The root is saved at the moment when the entire Patricia trie is downloaded.
2. We also clearly state that the node transitions to the full sync.
3. When you see the block being processed, then you are in the full sync and the newly arrived block is being
   calculated.
4. Every two minutes you will see a summary of connected peers with their client version, IP address, highest synced
   block, and data download speeds.

!GÃ¶rli full sync logs

Also, every now and then, a peer report will appear like below:

![](</img/Screenshot2022-08-10205144.png>)

1. First bracket is for Allocated contexts. It has possible values of H for Headers, B for Bodies, R for
   Receipts, N for State, S for Snap, and W for Witness.
2. Second bracket is for Sleeping contexts. It has possible values of H for Headers, B for Bodies, R for
   Receipts, N for State, S for Snap, and W for Witness.
3. Third bracket contains Peer Info.
4. Fourth bracket is for Speeds as Follows:
     Latency
     Headers Transfer
     Bodies Transfer
     Receipts Transfer
     Node Data Transfer
     Snap Ranges Transfer
5. Fifth bracket is for Client Info like Client Name, Client Version, Operating System and Language Version.

performance-tuning.md:

---
title: Performance tuning
sidebarposition: 7

By default, Nethermind is configured for general use cases that fit well for most users. However, to improve various aspects of Nethermind performance, there are options for different subsystems that can be configured for your specific needs.

Peer discovery

To connect to the Ethereum network, Nethermind needs to maintain connections to other clients. The number of connections can be configured with --Network.MaxActivePeers <value>. The default value depends on the network. Increasing this number may reduce syncing time, while reducing this number may help with attestation performance.
Also, you can increase the rate at which a new connection is established with --Network.MaxOutgoingConnectPerSec <value>. The default value is 20 while 50 would be a reasonable higher value. This tends to reduce the snap sync time; however, some ISPs may throttle your Internet connection if you set this value too high.

Also, some WiFi routers may hang if the value is set too high.

Port forwarding

While port forwarding is not strictly required, it helps significantly with finding peers and is essential for the network's overall health. The exact steps for port forwarding highly depend on your environment, router, and ISP. For most home configurations, automatic port forwarding can be turned on with --Network.EnableUPnP true.
Some ISPs are more restrictive and do not support port forwarding and/or utilize provider-level NAT. In such cases, your best option is to use a VPN that supports port forwarding. Keep in mind that consensus clients need a separate port forwarding.

Sync time

On the Ethereum mainnet, most of the syncing time is split into three phases: snap sync, old bodies, and old receipts. Strictly speaking, there are also fast sync, full sync, and state sync phases. However, they usually complete in less than a minute, with state sync usually taking up to 3 minutes.

At the moment, the best test case sync time is 1 hour 50 minutes for all phases with the following configuration:

- CPU: AMD Ryzen 9 7950X
- Memory: 128GB RAM
- Storage: Intel Optane SSD 905P Series 900GB
- Network: 1 Gbps Internet with TorGuard VPN with WireGuard protocol. Both execution and consensus clients port forwarding are set up manually.
- Command line options:

```
    --Network.EnableUPnP true
    --Network.MaxOutgoingConnectPerSec 50
    --Network.ProcessingThreadCount 32
    --Sync.TuneDbMode HeavyWrite
```

Snap sync

Snap sync is the process of downloading the Ethereum state tree. After it is complete, and after the state sync phase, Nethermind can process and follow the chain. The fastest tested snap sync and state sync time is 25 minutes.
This phase is the most I/O-intensive sync phase, and therefore, assuming a fast internet, the sync time highly depends on your SSD's write speed. Remember that most SSDs only advertise peak write speed, usually above 5GB/s. However, they tend to slow down significantly to around 0.5GB/s (or even less for a QLC SSD) after a few seconds. Therefore, look for SSDs with high sustained write speed.

Also, ensuring your SSD is sufficiently cooled to prevent thermal throttling is essential. This is often overlooked as most workloads rarely stress SSD as much; however, to reduce sync time, Nethermind will utilize your SSD to its limit. If, for whatever reason, you need to minimize the I/O load, you can specify a rate limit with --Db.MaxBytesPerSec 1000000000.

Nethermind temporarily changes the database configuration during sync to optimize it for writing, notably the option --Sync.TuneDbMode HeavyWrite is turned on by default. On some systems with slow SSDs, the use of the option --Sync.TuneDbMode AggressiveHeavyWrite may boost. Also, the option --Sync.TuneDbMode DisableCompaction can be used to disable compaction altogether. This is likely faster for systems using entry-level NVMe SSDs and is also useful to extend the lifespan of your SSD as it provides the lowest total writes possible. However, it uses about 3GB of extra memory during snap sync. The state sync phase may appear to hang for about 10 minutes as the whole database

compacts for the first time after snap sync.

If you are running on a VPS with artificially capped IOPS, or you are using SATA SSD (which is highly not recommended), increasing the state DB block size with --Db.StateDbBlockSize 16384 may help to reduce snap sync time. However, this negatively affects block processing time. An alternative is to turn on compaction readahead with --Db.CompactionReadAhead 128000; however, this may take up a few extra GB of memory depending on the readahead value.

Old bodies and receipts

Old bodies and old receipts are the process of downloading block bodies and receipts. This is required for some RPC methods, such as ethgetLogs, and for consensus clients to work correctly. If you don't need them, skip this phase with

```
--Sync.DownloadBodiesInFastSync false
--Sync.DownloadReceiptsInFastSync false
--Sync.NonValidatorNode true
```

Old bodies and receipts are mainly limited by your Internet connection. With a 1Gbps connection, they consume around 250MB/s and 500MB/s of writes, respectively, which is generally reasonable for most PCIE SSDs. On older systems or VPS with low single thread performance and high Internet speed, the block body deserialization may be a bottleneck, in which case, you can increase the number of network processing threads with --Network.ProcessingThreadCount 32. However, this may impact block processing time.

Block processing time and attestation

Block processing time is limited mainly by SSD performance. Strictly speaking, it's not the IOPS that matters, but the response time. Nevertheless, the IOPS is a good approximation as most SSDs don't advertise the response time.
To help further reduce reads from SSD, Nethermind has multiple levels of caching, which is tuned by the memory hint option --Init.MemoryHint 2000000000. If you are running a system with more than 16GB of memory, it is highly recommended to increase this value. In-memory pruning (turned on by default) also improves block processing time.

It is also possible to disable compression of the state DB with --Db.StateDbDisableCompression true that improves block processing time by 3% to 5% but increases disk space usage correspondingly. Block processing is susceptible to the number of peers connected. Therefore, after the node is synced, it makes sense to reduce the number of peers with --Network.MaxActivePeers 20.

private-networks.md:

```
---
title: Private networks
description: Use Kurtosis to deploy a private Ethereum devnet with Nethermind and any consensus client at any scale you need, wherever you need it.
sidebarposition: 8
---
```

This guide will walk you through using Kurtosis ethereum-package to spin up a private, proof-of-stake (PoS) Ethereum devnet with three full Ethereum nodes locally over Docker. At the end of the guide, you will learn how to scale up your testnet on Kubernetes as well as enable optional services for your local testnet, such as network observability tools (e.g., Grafana, Prometheus) and Flashbot's mev-boost infrastructure to simulate MEV workflows.

Step 1: Prerequisites

Before you begin, ensure you have Kurtosis CLI and Docker installed. The ethereum-package is a Kurtosis environment definition known as a package. For more info about Kurtosis, see the Kurtosis docs.

Step 2: Configure your network

Next, in your home directory, create a file with the name networkparams.json and populate it with the following contents:

```
json title="networkparams.json"
{
  "participants": [
    {
      "elclienttype": "nethermind",
      "elclientimage": "nethermind/nethermind:latest",
      "clclienttype": "lighthouse",
      "clclientimage": "sigp/lighthouse:latest",
      "count": 1
    },
    {
      "elclienttype": "nethermind",
      "elclientimage": "nethermind/nethermind:latest",
      "clclienttype": "teku",
      "clclientimage": "consensys/teku:latest",
      "count": 1
    },
    {
      "elclienttype": "nethermind",
      "elclientimage": "nethermind/nethermind:latest",
      "clclienttype": "lodestar",
      "clclientimage": "chainsafe/lodestar:next",
      "count": 1
    }
  ],
  "mevtype": "None",
  "launchadditionalservices": false
}
```

As you can see above, you have effectively created a network configuration file that Kurtosis will use to pass in the necessary parameters at runtime for your network. Notice that the participant key describes the execution and consensus client pairing desired for each full node and how many nodes of that type to instantiate.

There are many other configurations and flags you can use, including metrics and observability tools (e.g., Grafana, Prometheus, etc). For all supported options, see the ethereum-package configuration.

Step 3: Deploy

Finally, once you have saved the networkparams.json file, it is time to deploy the private net:

```
bash
kurtosis run github.com/ethpandaops/ethereum-package "$(cat
/networkparams.json)"
```

Kurtosis will use the ethereum-package environment definition and your custom network configuration (defined in networkparams.json) to spin up your network.

Kurtosis will first spin up an [enclave][enclaves] (i.e., an ephemeral, isolated environment) and begin to configure and instantiate the nodes in your network. In the end, Kurtosis will print the services running in your enclave that form your private testnet alongside all the container ports and files that were generated and used to start up the private testnet.

Here is a sample output:

```
INFO[2023-09-01T16:10:45-04:00]
===============================================
INFO[2023-09-01T16:10:45-04:00] ||          Created enclave: timid-knoll
||
INFO[2023-09-01T16:10:45-04:00]
===============================================
Name:            timid-knoll
UUID:            939dfb5d59b0
Status:          RUNNING
Creation Time:   Fri, 01 Sep 2023 16:08:57 EDT

========================================== Files Artifacts
==========================================
UUID           Name
a876b06035b7   1-lighthouse-nethermind-0-63
87955ef69845   2-teku-nethermind-64-127
4f77377da494   3-lodestar-nethermind-128-191
9734313101e3   cl-genesis-data
4164ed5c594c   el-genesis-data
a49a3d2774b5   genesis-generation-config-cl
16fcc4f96236   genesis-generation-config-el
5fc72346f646   geth-prefunded-keys
96ae153a0b51   prysm-password

============================================ User Services
============================================
UUID           Name                                       Ports
Status
f369802ad2ae   cl-1-lighthouse-nethermind                  http: 4000/tcp ->
http://127.0.0.1:49894      RUNNING
                                                          metrics: 5054/tcp ->
http://127.0.0.1:49892
                                                          tcp-discovery:
9000/tcp -> 127.0.0.1:49893
                                                          udp-discovery:
9000/udp -> 127.0.0.1:64949
5e14eb26ef45   cl-1-lighthouse-nethermind-validator       http: 5042/tcp ->
127.0.0.1:49895             RUNNING
                                                          metrics: 5064/tcp ->
http://127.0.0.1:49896
fed533d0e143   cl-2-teku-nethermind                       http: 4000/tcp ->
127.0.0.1:49899             RUNNING
                                                          metrics: 8008/tcp ->
127.0.0.1:49897
                                                          tcp-discovery:
9000/tcp -> 127.0.0.1:49898
                                                          udp-discovery:
9000/udp -> 127.0.0.1:55521
69cd832de246   cl-3-lodestar-nethermind                   http: 4000/tcp ->
127.0.0.1:49903             RUNNING
                                                          metrics: 8008/tcp ->
127.0.0.1:49901
                                                          tcp-discovery:
9000/tcp -> 127.0.0.1:49902
                                                          udp-discovery:
```

```
9000/udp -> 127.0.0.1:50507
75e3eec0c7d1   cl-3-lodestar-nethermind-validator         metrics: 8008/tcp ->
127.0.0.1:49904          RUNNING
e10c3f07e0e0   el-1-nethermind-lighthouse                engine-rpc: 8551/tcp -
> 127.0.0.1:49872       RUNNING
                                                         rpc: 8545/tcp ->
127.0.0.1:49870
                                                         tcp-discovery:
30303/tcp -> 127.0.0.1:49869
                                                         udp-discovery:
30303/udp -> 127.0.0.1:64508
                                                         ws: 8546/tcp ->
127.0.0.1:49871
c6a28d3136fe   el-2-nethermind-teku                      engine-rpc: 8551/tcp -
> 127.0.0.1:49873       RUNNING
                                                         rpc: 8545/tcp ->
127.0.0.1:49875
                                                         tcp-discovery:
30303/tcp -> 127.0.0.1:49874
                                                         udp-discovery:
30303/udp -> 127.0.0.1:52495
                                                         ws: 8546/tcp ->
127.0.0.1:49876
2fae3b3c41d3   el-3-nethermind-lodestar                  engine-rpc: 8551/tcp -
> 127.0.0.1:49890       RUNNING
                                                         rpc: 8545/tcp ->
127.0.0.1:49888
                                                         tcp-discovery:
30303/tcp -> 127.0.0.1:49891
                                                         udp-discovery:
30303/udp -> 127.0.0.1:62119
                                                         ws: 8546/tcp ->
127.0.0.1:49889
403cafe8416e   prelaunch-data-generator-cl-genesis-data   <none>
RUNNING
ebea71008cf4   prelaunch-data-generator-el-genesis-data   <none>
RUNNING
```

And that is it! You now have a 3-node, private Ethereum devnet with Nethermind/Lodestar, Nethermind/Teku, and Nethermind/Lighthouse execution and consensus client combinations.

Notice how, at the end, Kurtosis will print out the contents of your enclave, which includes both the various files artifacts and services that form your network. Kurtosis also maps the container ports to ephemeral local ports on your machine.

Genesis data was generated using this genesis-generator to bootstrap the execution and consensus clients for each node. The end result will be a private testnet with nodes deployed as Docker containers in an ephemeral, isolated environment on your machine called an [enclave][enclaves].

Kurtosis packages are modular, reproducible, and will work over Docker or Kubernetes. Read on to learn about additional services and configurations you may want to add to your private network.

Step 4: Optional workflows

This section briefly covers some optional configurations for your private devnet that are commonly used for validating and testing node-level behavior.

Simulating MEV workflows with mev-boost

The ethereum-package can simulate out-of-protocol Proposer Builder Separation (PBS) workflows using Flashbot's mev-boost infrastructure. With a single flag, you can configure your network's validators to be instantiated with mev-boost and be registered with a relayer to receive payloads from builders.

To enable this in your networkparams.json file, set "mevtype": to "full" or "mock".

To learn more about how the mev-boost infrastructure works with your private network, check out this guide.

Observability tools

The ethereum-package comes out of the box with a few observability tools, including:

- A Grafana and Prometheus instance
- A beacon metrics gazer service to collect network-wide participation metrics
- A JSON-RPC Snooper to log responses & requests between the execution engine API and the consensus client

To add your own custom Grafana dashboard template, fork the ethereum-package repository and add your configuration here.

Deploying on Kubernetes

As mentioned earlier, Kurtosis packages (i.e. environment definitions) are portable and will work the same way over Docker or on Kubernetes. Should you require a larger scale devnet, Kurtosis can deploy any package, including the ethereum-package on Kubernetes, see Running Kurtosis in Kubernetes.

Questions and feedback

If you need help with your Nethermind full node in the private devnet, please don't hesitate to contact the Kurtosis team on GitHub or Discord.

[enclaves]: https://docs.kurtosis.com/concepts-reference/enclaves

pruning.md:

---
title: Pruning
sidebarposition: 6

Overview

Pruning pertains to eliminating or cleaning obsolete historical data to optimize disk space. Clients are responsible for maintaining the world state, which comprises a database that portrays the current Ethereum network status. The world state encompasses accounts, contracts, and other information.

Pruning aims to reduce disk requirements by storing only the current world state and removing historical data. This differs from archive nodes that retain complete transaction and state history. Pruning is helpful for users who don't require historical data and prefer to interact solely with the current state of the network. However, pruning may limit the client's ability to fulfill requests that depend on historical information. Nethermind provides two kinds of pruning â “ full pruning and in-memory pruning; both are enabled by default, also called hybrid pruning.

How it works

During synchronization using the snap sync method, Nethermind produces a local

copy of the Ethereum network state. Although this size increases by around 30 GB each week, specific historical data is retained that is not necessary for node operation or to maintain the current Ethereum state. For a detailed description of the disk usage usage, check out the database size.

When full pruning is activated and initiated, a thorough examination of the entire state tree is conducted to determine which data is no longer required and can be treated as historical. It then determines which information corresponds to the current state and duplicates it alongside the existing version. During verification of each node in the state, the new pruned state replaces the previous one. Once the verifier confirms everything is functioning correctly, the old state database is eliminated, resulting in significant savings in disk space. As a result, the size of the database will be close to its initial size again.

In-memory pruning is a continuous process that occurs under regular operation. Instead of saving a new state on each block, Nethermind will keep it in memory until a certain threshold is reached. At that point, Nethermind will only store data required by the newer state and discard unnecessary ones. This significantly reduces the total amount of data written while improving block processing performance. In-memory pruning is independent of full pruning.

Preparation for full pruning

Because full pruning is executed while the node is connected to the network, it can affect the node's performance.
The process can consume significant memory, CPU, and disk resources, impacting block processing time.
This, in turn, can lead to reduced rewards for validator setups, particularly for proof-of-stake chains.

Currently, full pruning takes between 20 to 30 hours to complete, although the duration may vary based on the hardware configuration. At least 200 GB of storage is required to ensure pruning runs smoothly. Nevertheless, it's recommended to have 300 GB or more.

:::warning Important
Do not turn on full pruning on an archive node, as these are two opposing features. Archive nodes are designed to store
complete historical data, whereas full pruning eliminates it. Use the option --Pruning.Mode None to ensure that pruning is turned off completely.
:::

Configuring full pruning

As a very first point, check out the pruning configuration options.\
To activate full pruning, use either the --Pruning.Mode Hybrid or --Pruning.Mode Full command line options.

:::info
Setting --Pruning.Mode Hybrid enables both InMemory and Full modes. The InMemory mode helps the node storage grow slower than the Full mode. Because of this, full pruning is executed less frequently, promoting
healthier disk operation. Since full pruning is hardware-intensive, this configuration also benefits attestation results.
:::

The next step is to determine the trigger conditions for full pruning.
Currently, there are 3 options available:

- Manual
- State database size threshold
- Remaining storage space threshold

Manual

Manual mode triggers full pruning only upon request, providing complete control. To configure this mode, use the following options: --Pruning.Mode Hybrid --Pruning.FullPruningTrigger Manual.

Pruning can also be triggered using the adminprune JSON-RPC method. Here's how to configure it:

- Add the admin namespace to --JsonRpc.EnabledModules. For instance,\
  --JsonRpc.EnabledModules [eth,net,...,admin]
- Create a separate port for admin namespace only:\
  --JsonRpc.AdditionalRpcUrls http://localhost:8555|http|admin

Restart the client, and if everything is configured correctly, you should be able to call the adminprune method, and full pruning should start.

:::info
One potential disadvantage of the manual mode is that if full pruning is not triggered on time, it cannot be performed later because of insufficient disk space. In such cases, the only option to free up the disk space is to resync the node from scratch.
:::

State database size threshold {#state-db-threshold}

This mode triggers full pruning automatically when the state database reaches the specified size.
To enable this mode, use the following options: --Pruning.Mode Hybrid --Pruning.FullPruningTrigger StateDbSize --Pruning.FullPruningThresholdMb 256000, where the value of Pruning.FullPruningThresholdMb should be set based on your requirements.

The above configuration triggers full pruning automatically whenever the state database size exceeds 256,000
MB (250 GB). Assuming the state database has initially around 160 GB, pruning will be triggered when the database size grows by more than 90 GB.

:::tip
To avoid unexpected behavior and ensure that full pruning can be completed in full, it's recommended to set the threshold value to trigger pruning before the free disk space drops below 250 GB as a minimum. This ensures that sufficient free disk space is available for the pruning process.
:::

Remaining storage space threshold {#storage-left-threshold}

:::note
This is the recommended approach as it ensures that pruning is executed on time.
:::

This mode triggers full pruning when the storage space reaches the specified minimum. To enable this mode, use the following options: --Pruning.Mode Hybrid --Pruning.FullPruningTrigger VolumeFreeSpace --Pruning.FullPruningThresholdMb 256000, where the value of Pruning.FullPruningThresholdMb should be set based on your requirements. However, it should not be set below the default value of 256000.

The above configuration triggers full pruning whenever free disk space drops to 256,000 MB (250 GB) or below. This ensures that pruning is invoked as infrequently as possible while also ensuring that sufficient free storage is always available to trigger it.

:::warning

It's recommended not to set the value below 250 GB for stability reasons. In reality, full pruning should require approximately the same amount of storage as a fresh state database (around 160 GB). Still, it also needs a threshold for processing and other operations since it creates a copy of the existing state database. Therefore, it's essential to maintain a minimum amount of storage to ensure a robust pruning performance.
:::

Monitoring progress

When full pruning is triggered correctly, the corresponding messages appear in the Nethermind logs.

The very first ones should be:

```
Full Pruning Ready to start: pruning garbage before state <block number> with root <hash>.
WARN: Full Pruning Started on root hash <hash>: do not close the node until finished or progress will be lost.
```

From that moment, ensure that no restarts will be performed on Nethermind to ensure that full pruning runs correctly.
After a few minutes first logs with progress would start to appear. For instance:

```
Full Pruning In Progress: 00:00:57.0603307 1.00 mln nodes mirrored.
Full Pruning In Progress: 00:01:40.3677103 2.00 mln nodes mirrored.
Full Pruning In Progress: 00:02:25.6437030 3.00 mln nodes mirrored.
```

Pruning may take even more than 30 hours, depending on hardware configuration.

When pruning is completed, there is a log message similar the following:

```
Full Pruning Finished: 15:25:59.1620756 1,560.29 mln nodes mirrored.
```

As you may notice, it took around 15 hours in this example.

:::info
Since the amount of mirrored nodes is not a static value, providing a simple progress indicator in percentage is impossible. Therefore, the approximate value given should be used to determine when full pruning is expected to be completed.
:::

Additional settings

Memory budget

The Pruning.FullPruningMemoryBudgetMb configuration option controls the memory budget allocated for the trie visit during
the full pruning process. During pruning, pending nodes are queued to a pool of nodes whose size is determined by this value. This allows multiple nodes to share a single I/O. By increasing this value, the required read IOP per second can be significantly reduced, resulting in a faster full pruning operation. However, this improvement comes at the expense of increased memory usage.

Assuming your system has 64GB of RAM, with Nethermind, the consensus client, and system expenses consuming 20GB,
you need to determine the maximum value for Pruning.FullPruningMemoryBudgetMb

while ensuring the system remains stable and respects the given limit. In this case, you have 44 GB (64 GB - 20 GB) of available memory for increasing the Pruning.FullPruningMemoryBudgetMb value. To calculate the maximum value in MB, multiply the available memory by 1024: 44 GB 1024 = 45,056 MB

:::info
Because of the workload on Ethereum Mainnet, setting the value of Pruning.FullPruningMemoryBudgetMb higher than 16 GB may not provide any additional performance benefits.
:::

Please note that this example assumes the remaining 44 GB of memory is sufficient for the full pruning memory budget.
Depending on the specific use case and system requirements, it may be necessary to adjust the value further to optimize performance and resource usage.

Pruning completion behavior

The Pruning.FullPruningCompletionBehavior configuration option determines Nethermind's behavior after full pruning is completed. By default, Nethermind will continue to progress as usual. However, if a user wishes to shut down the node after pruning, there are three options available:

- None: No action taken
- ShutdownOnSuccess: Nethermind shuts down if pruning succeeds
- AlwaysShutdown: Nethermind shuts down once pruning completes, regardless of whether it succeeds or fails

Number of pruning concurrent tasks

The Pruning.FullPruningMaxDegreeOfParallelism configuration option determines the number of parallel tasks/threads that can be used by pruning:

- -1: uses the number of logical processors
- 0: uses 25% of logical processors
- 1: runs on a single thread

The recommended value depends on the type of node being used. If the node needs to be responsive (e.g., RPC or validator), using a value below the number of logical processors is recommended. The default value is recommended if the node doesn't have many other responsibilities but needs to follow the chain without delays and produce live logs reliably. If the node doesn't need to be responsive, has very fast I/O (such as NVMe), and the shortest pruning time is desired, this can be set to 2â “3 times the number of logical processors.

In-memory cache size

The Pruning.CacheMb configuration option determines the size, in MB, of the memory pool of nodes used for in-memory pruning. The default value is 1024. Increasing this value can help reduce the rate at which the state database grows.

Side notes

For pruning, keep in mind the following:

- Full pruning is a cumbersome task, but it's performed in the background, so the node continues progressing and following the chain.
- The process' heaviness may affect the effectiveness of the validator rewards. Still, since it's executed only once every few months, it shouldn't have a significant impact on overall results (we've xperienced approximately 5â “10% loss of rewards during full pruning).
- Ensure that your storage has at least 250 GB of free space after syncing the node. Otherwise, full pruning will never complete successfully.

- Several things can be done to reduce the size of the database after syncing: setting Sync.AncientBodiesBarrier and Sync.AncientReceiptsBarrier to a proper value higher than 0, using a consensus client that requires less storage, and setting logs to the lowest level to avoid log spamming.

security.md:

---
title: Security
sidebarposition: 3

General security considerations

:::tip
Although Nethermind is thoroughly tested, the more popular it becomes, the more likely it will be a target of client-specific attacks. Generally, we recommend you always consider running backup client nodes from another developer for any critical operations.
:::

:::warning
Enable only the JSON-RPC namespaces you absolutely need. This is particularly important for namespaces like admin and debug, as they can be exploited to get elevated access to your node or for DOS attacks.
:::

:::danger
The private key the node id is derived from is stored on the disk as plaintext.
:::

Networking security

These rules are highly recommended to be applied to your firewall:

- Block all traffic to the port 8545, or whatever port is defined for JSON-RPC interface, except for traffic from explicitly defined trusted sources.
- Allow traffic to the TCP port 30303 or whatever port is defined for P2P communication. This allows the node to connect to peers.
- Allow traffic to the UDP port 30303 or whatever port ia defined for P2P communication. This allows node discovery.

sync.md:

---
title: Sync
sidebarposition: 2
---

import Tabs from "@theme/Tabs";
import TabItem from "@theme/TabItem";

:::warning
This article requires a revision.
:::

There are three main synchronization modes

 snap sync
       the fastest way to sync to the network (syncs to mainnet in \3 hours)
       downloads only the latest state, headers, and optionally bodies and

receipts
 fast sync
       slower than snap sync
       useful on nethermind-only chains (like Gnosis), where snap sync is not available for now
       downloads only the latest state, headers, and optionally bodies and receipts
 archive sync
       heavy historical sync verifying all the transactions and keeping all the historical state
       you can run it like this ./Nethermind.Runner --config mainnetarchive

Sync Mode	Disk Space needed	Full current state	Full current and all historical states	Can sync a full archive node from this	Time to sync	Time to RPC
archive	\12TB	YES	YES	YES	\3 weeks	\3 weeks
snap/fast sync with all bodies and receipts	\800GB	YES	NO	YES	\20 hours	\20 hours
default snap/fast sync (with barriers set to support Eth2)	\500GB	YES	NO	NO	\11 hours	\11 hours
snap/fast sync without receipts	\450GB	YES	NO	YES	\12 hours	\12 hours
snap/fast sync without bodies and receipts	\200GB	YES	NO	NO	\9 hours	\9 hours

Snap Sync

Snap sync allows a node to perform the initial synchronization and download of Ethereumâ ™s state up to 10 times faster
than using fast sync.

How to Enable

Snap sync is enabled by default for majority of networks. To make sure, check ifSnapSync is set to true in the Sync
module of your .cfg file

```
"Sync": {
      "SnapSync": true
}
```

> IMPORTANT: Do not enable snap sync on a previously synced node. Only use when syncing to the network for the first
> time.
>
> IMPORTANT: Do not remove other configurations from Sync module. Just add "SnapSync": true, e.g.
>
>
> "Sync": {
>   "SnapSync": true,
>   "PivotNumber": 15011000,
>   "PivotHash":

"0x46c838d02d5fa5bc070080ed7965da1d888f6eb1797045365407c7011280af56",
>   "PivotTotalDifficulty": "52367203434576253689712",
>   "FastBlocks": true
> }
>

Snap Sync vs Other Sync Modes

More than 12TB of storage is needed today to run a full archive node â ” one that stores all the state since genesis.
Because of that, setting up an archive node can take days or even weeks. Fast Sync can still take more than 24 hours on
the fast machine and download about 90GB state data. With Snap Sync, sync time is reduced to 2-3h with a download of
about 30GB.

This reduction in sync time and download size has to do with the specific way in which Ethereumâ ™s state is stored in a
node: Merkle trees.

![](</img/Untitled(1).png>)

With Fast Sync, a node downloads the headers of each block and retrieves all the nodes beneath it until it reaches the
leaves. By contrast, Snap Sync only downloads the leaf nodes, generating the remaining nodes locally which saves time
and packets downloaded.

Current limitations and future development

For now Snap Sync on the Nethermind client can only download the Ethereum state but not serve it to other clients - snap
serving development is in progress, expected late 2023/early 2024.

The only Ethereum client that supports serving Snap Sync requests is Geth, so only networks supported by Geth can be
synced using that method: Mainnet, Goerli, Sepolia.

Fast Sync

After completing the fast sync your node will have the ability to answer questions like 'what is my account
balance now', 'how many XYZ tokens SomeExchange holds at the moment'.

Fast sync has multiple stages. Nethermind uses a pivot block number to improve fast sync performance.
The pivot block data is automatically updated after initialization of the client and consists of
the block number, block hash and block total difficulty (please note that total difficulty is different
than difficulty). Before synchronizing state data Nethermind synchronizes in two directions - backwards
from pivot block to 0 for headers and forward to the head of the chain for headers, blocks and receipts. Forward sync
might be very slow (5 - 50 bps) so having fresh pivot block is very important - it is guaranteed by recently implemented
auto-update.

After downloading the block data Nethermind will start state sync (downloading the latest state trie nodes). We are
providing an estimate for the download size and progress but the real value may be different than the estimate (
especially if you are using an old version of Nethermind as we sometimes manually adjust the estimator based on our

observations of the chain growth rate). Because of this sometimes your sync may continue even when it shows \100%
finished. The other important component is the speed of your sync - if your IO / network / file system causes the state
sync to go much slower than around 1.5MB per second on average then you will start downloading some parts of the trie
over and over again. In such cases you may be surprised to see something like 58000MB / 53000MB (100%). It means that
you downloaded around 5GB of data that is no longer needed. If your sync is very slow (extended beyond two days) then
very likely your setup cannot catch up with the chain progress.

After the state sync finishes you will see the 'Processed...' messages like in archive sync - it means that your node
is in sync and is processing the latest blocks.

Mainnet sync, at the time of writing (December 2020), takes around 8 hours on an UpCloud 16GB RAM 8 CPU $40 VM (and then
syncs receipts and bodies in the background if you enabled them in the configuration). Goerli sync should take around 40
minutes.

!Fast sync logs example for mainnet..png>)

State sync log messages have multiple values displayed. First dd.HH:mm:ss total state sync time is displayed, followed
by an estimated sync progress (percentage of total database data synced), then the current download speed is displayed (
there will be times when it will slow down to 0 or single digit numbers, especially towards the end of the sync). In
general 6 hours sync times shown on screenshots are with around 2000 kB/s (kilobytes per second) average sync rate. You
can calculate it in the example as \45GB / (2MB/s) \ 22500 seconds \6.25 hours. We also display the number of state
accounts synced, number of trie nodes synced and a diagnostic message in the format of \[number\of\pending\requests]
.\[average time spent in response handler displayed as milliseconds]. So 5.6.20ms means that we are awaiting for
responses to 5 requests that have been sent to peers and the average time it takes us to process a single response is
6.20ms. The response handling times will differ depending on how many trie nodes are already cached (so they will be
significantly slower for a while after the node restart when cache has to be rebuilt) and based on how fast the database
IO is (SSD vs NVMe vs cloud drives). For a reasonable sync time you probably should expect these values to be below
15ms (but they may be as high as 700ms for a while after restarting the node).

A single restart of the node during the fast sync may extend the sync time by up to two hours because the node has to
rebuild the caches by reading millions of values from the database.

At the last stages of the sync the node will be repeatedly displaying the branch sync progress and changing the block
number to which it tries to catch up. This stage should take between 30 minutes and two hours. If it lasts much more
then it is possible that you will not be able to catch up with the network progress.

One of the best indicators that you are close to be synced is combined \100% state size progress and nearly 100% branch
sync progress.

![](</img/image(64).png>)

Archive Sync

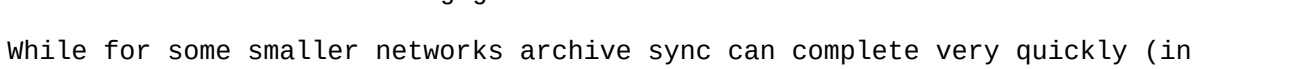
Archive sync is the 'heaviest' and slowest sync mode but allows to ask questions like 'what was the balance of my
account 2 years ago?', 'how many XYZ token were held in SomeExchange custody in 2017?'.

We have prepared default archive sync configurations and they can be launched from Nethermind Launcher (just choose the
archive options) or by simply loading appropriate config when launching ./Nethermind.Runner --config mainnetarchive

./Nethermind.Runner --config goerliarchive

While for some smaller networks archive sync can complete very quickly (in minutes or hours) mainnet sync would take 2 -
6 weeks depending on the speed of your IO (whether you use SSD or NVMe or depending on the cloud provider IOPS).
Database size in archive sync is the biggest from all modes as you will store all the historical data.

!Example of the archive sync logs.png>)



Explanation of some data in the logs:

 at the beginning you may see a 'Waiting for peers...' message while the node is trying to discover nodes that it can
  sync with.
 'Downloaded 1234/8000000' shows the number of unprocessed blocks (with transactions) downloaded from the network.
  For mainnetthis value may be slower than processing at first but very quickly you will see blocks being downloaded
  much faster than processed. Empty blocks can be as small as 512 bytes (just headers without transactions) and full
  blocks with heavy transactions can reach a few hundred kilobytes. We display both current download speed (calculated
  in the last second) and average (total) speed since starting the node.
 'Processed ...' displays the blocks that have been processed by the EVM. The first number shows the current head
  block number, then you can see mgasps (million gas per second) - current and total, then tps (transactions per
  second) - current and total, bps (blocks per second). Then recv queue (transactions signature public key recovery
  queue), proc queue (processor queue). Both recovery queue and processor queue are designed so when too many blocks
  are waiting for processing then only their hashes are kept in memory and remaining data are stored in the database.
  Thus, the queues numbers that you can see will be capped by some number.
 'Cache for epoch...' informs about ethash cache needed for block seal verification (only on mainnet
  and ropsten). Caches will be calculated every 30000 blocks (length of an epoch) but can also be calculated for the
  latest blocks that are being broadcast on the network.
 After the archive sync finishes you will see the 'Processed...' message appearing on average every 15 seconds when
  the new block is processed.
 mgasps, tps, bps values should not be treated as comparable as they may differ massively on different parts of
  the chain. For example when blocks are empty you may see very high bps values

with very low (or even zero) tps
  and mgasps values as there are no transactions and no gas for EVM processing and blocks are very light. On the other
  hand when blocks are filled with very heavy transactions then bps might be very low while mgasps will be very
  high. It is even possible that you will see a lot of very light transactions where tps will be high while bps
  and mgasps will be average.

Sync time

Sync time heavily depends on the hardware used for the node, network speed, and peering. We are constantly pursuing to make it as fast as possible. Below is a brief on how the sync time looks on different machines and various chains (tested with Nethermind v1.21.0).

<Tabs>
<TabItem value="highend-vm" label="High-end VM">

Hardware configuration:

- Cloud provider: Akamai (formerly Linode)
- CPU: AMD EPYC 7601, 16 vCPU
- Memory: 64 GB
- Storage: 1.2 TB, 40k IOPS

<Tabs groupId="network">
<TabItem value="mainnet" label="Mainnet">

The high-level data on major sync milestones:

- Attestation time: 2h 3m
- Full sync time: 7h 3m

The detailed breakdown of sync stages:

- Snap sync phase 1: 1h 58m
- State sync: 4m
- Old headers: 1h 27m
- Old bodies: 1h 55m
- Old receipts: 3h 2m

</TabItem>
<TabItem value="goerli" label="Goerli">

The high-level data on major sync milestones:

- Attestation time: 2h 49m
- Full sync time: 4h 58m

The detailed breakdown of sync stages:

- Snap sync phase 1: 2h 49m
- State sync: 0.5m
- Old headers: 11m
- Old bodies: 1h 2m
- Old receipts: 1h 5m

</TabItem>
<TabItem value="sepolia" label="Sepolia">

The high-level data on major sync milestones:

- Attestation time: 8m

- Full sync time: 1h 9m

The detailed breakdown of sync stages:

- Snap sync phase 1: 8m
- State sync: 0.3m
- Old headers: 12m
- Old bodies: 21m
- Old receipts: 22m

</TabItem>
<TabItem value="gnosis" label="Gnosis">

The high-level data on major sync milestones:

- Attestation time: 13h 40m
- Full sync time: 17h 17m

The detailed breakdown of sync stages:

- State sync: 13h 40m
- Old headers: 1h 46m
- Old bodies: 1h 31m
- Old receipts: 2h 3m

</TabItem>
<TabItem value="chiado" label="Chiado">

The high-level data on major sync milestones:

- Attestation time: 20m
- Full sync time: 40m

The detailed breakdown of sync stages:

- State sync: 20m
- Old headers: 11m
- Old bodies: 8m
- Old receipts: 10m

</TabItem>
</Tabs>
</TabItem>
<TabItem value="midend-vm" label="Mid-end VM">

Hardware configuration:

- Cloud provider: AWS
- c7g.2xlarge: 8 vCPU, 16 GiB memory
- Storage: 1 TB, 10k IOPS

<Tabs groupId="network">
<TabItem value="mainnet" label="Mainnet">

The high-level data on major sync milestones:

- Attestation time: 5h 55m
- Full sync time: 12h 37m

The detailed breakdown of sync stages:

- Snap sync phase 1: 4h 35m
- State sync: 1h 20m
- Old headers: 1h 43m

- Old bodies: 2h 13m
- Old receipts: 4h 28m

</TabItem>
<TabItem value="goerli" label="Goerli">

The high-level data on major sync milestones:

- Attestation time: 1h 32m
- Full sync time: 4h 10m

The detailed breakdown of sync stages:

- Snap sync phase 1: 1h 19m
- State sync: 12m
- Old headers: 23m
- Old bodies: 49m
- Old receipts: 1h 35m

</TabItem>
<TabItem value="sepolia" label="Sepolia">

The high-level data on major sync milestones:

- Attestation time: 17m
- Full sync time: 1h 3m

The detailed breakdown of sync stages:

- Snap sync phase 1: 13m
- State sync: 4m
- Old headers: 15m
- Old bodies: 19m
- Old receipts: 29m

</TabItem>
<TabItem value="gnosis" label="Gnosis">

The high-level data on major sync milestones:

- Attestation time: 15h 54m
- Full sync time: 18h 28m

The detailed breakdown of sync stages:

- State sync: 15h 54m
- Old headers: 1h 4m
- Old bodies: 40m
- Old receipts: 1h 52m

</TabItem>
<TabItem value="chiado" label="Chiado">

The high-level data on major sync milestones:

- Attestation time: 13m
- Full sync time: 25m

The detailed breakdown of sync stages:

- State sync: 13m
- Old headers: 12m
- Old bodies: 5m
- Old receipts: 5m

```
</TabItem>
</Tabs>
</TabItem>
<TabItem value="oldspec-vm" label="Old-spec VM">

Hardware configuration:

- Cloud provider: Scaleway
- CPU: Intel Xeon Processor E5-2620 v2, 2 vCPU
- Memory: 192 GB
- Storage: 1 TB, 44k IOPS

<Tabs groupId="network">
<TabItem value="mainnet" label="Mainnet">

The high-level data on major sync milestones:

- Attestation time: 5h 55m
- Full sync time: 17h 1m

The detailed breakdown of sync stages:

- Snap sync phase 1: 4h 29m
- State sync: 25m
- Old headers: 1h 27m
- Old bodies: 3h 39m
- Old receipts: 8h 3m

</TabItem>
<TabItem value="goerli" label="Goerli">

The high-level data on major sync milestones:

- Attestation time: 1h 51m
- Full sync time: 5h 55m

The detailed breakdown of sync stages:

- Snap sync phase 1: 1h 40m
- State sync: 11m
- Old headers: 50m
- Old bodies: 1h 34m
- Old receipts: 2h 14m

</TabItem>
<TabItem value="sepolia" label="Sepolia">

The high-level data on major sync milestones:

- Attestation time: 16m
- Full sync time: 2h 9m

The detailed breakdown of sync stages:

- Snap sync phase 1: 15m
- State sync: 1m
- Old headers: 26m
- Old bodies: 45m
- Old receipts: 56m

</TabItem>
<TabItem value="gnosis" label="Gnosis">
```

The high-level data on major sync milestones:

- Attestation time: 15h 13m
- Full sync time: 17h 30m

The detailed breakdown of sync stages:

- State sync: 15h 13m
- Old headers: 3h 8m
- Old bodies: 50m
- Old receipts: 1h 25m

</TabItem>
<TabItem value="chiado" label="Chiado">

The high-level data on major sync milestones:

- Attestation time: 20m
- Full sync time: 40m

The detailed breakdown of sync stages:

- State sync: 4m
- Old headers: 1h 27m
- Old bodies: 1h 55m
- Old receipts: 3h 2m

</TabItem>
<TabItem value="energyweb" label="Energy Web">

The detailed breakdown of sync stages:

- State sync: 13h 7m
- Old headers: 2h 32m
- Old bodies: 51m
- Old receipts: 1h 11m
- Full sync time: 15h 20m

</TabItem>
<TabItem value="volta" label="Volta">

The detailed breakdown of sync stages:

- State sync: 14h 27m
- Old headers: 2h 42m
- Old bodies: 40m
- Old receipts: 58m
- Full sync time: 16h 10m

</TabItem>
</Tabs>
</TabItem>
</Tabs>

Resync a node from scratch

This guide explains how to resync a Nethermind node using the existing Pivot block or updating it to a more recent one.

Steps to Resync a Nethermind Node

1. Stop the Nethermind node: If your Nethermind node is currently running, stop it to ensure that no new data is
   being written to the database during the resync process.

2. Delete the existing database: Navigate to the Nethermind data directory. The location of this directory depends
   on how Nethermind was installed and your configuration settings. Inside the data directory, find the netherminddb
   folder and delete the mainnet subfolder to remove the existing database for the mainnet.
3. Update the configuration file (optional): If you want to change any configuration settings before resyncing the
   node, edit the mainnet.cfg file located in the Nethermind directory. For example, you might want to adjust the
   pruning settings or specify a different network.
4. Update the Pivot block (optional)\
   :::danger
   Only for versions before 1.19.0 where Auto-Pivot approach was introduced
   :::
      1. Using Etherscan: If you want to speed up the syncing process, you can update the Pivot block to a more recent
         one. To do this, find the Sync section in the mainnet.cfg file and update the PivotNumber and PivotHash
         fields to match a recent "finalized" block number and its corresponding hash. You can obtain this information from
         a block explorer such as Etherscan.\
         \
         Using block number 17165278 from Etherscan:

```
            {
                  "PivotNumber": 17165278,
                  "PivotHash":
"0xa665315efd923f3b11215feee09a9d3e13c5e6ee602fa19b642824682ec0a752"
            }
```

      2. Using Nethermind's GitHub: Alternatively, you can update the Pivot block by referring to
         the Nethermind's mainnet.cfg file on GitHub.
         The Pivot block is periodically bumped to the HEAD-8192 block of the mainnet chain. Copy the PivotNumber
         and PivotHash values from the GitHub file and update your local mainnet.cfg file accordingly.
5. Restart the Nethermind node: Start the Nethermind node again to initiate the resync process. The node will begin
   syncing from the existing Pivot block or the specified updated Pivot block, downloading and processing all the blocks
   in the blockchain.

To ensure that your Nethermind node is resyncing, you can monitor the logs for the node's progress. The logs will
display information about the block processing, synchronization status and OldHeaders being processed. By observing
the increasing block numbers and synchronization messages in the logs, you can confirm that the resync process is active
and working as expected.

```
> Old Headers 0 / 17154000 | queue 0 | current 0.00bps | total 0.00bps
>
> Old Headers 768 / 17154000 | queue 0 | current 766.07bps | total 762.49bps
>
> Beacon Headers from block 17154001 to block 17169722 | 960 / 15722 | queue
4992 | current 0.00bps | total
> 40622848.83bps
>
> Old Headers 9024 / 17154000 | queue 0 | current 576.40bps | total 1286.40bps
>
> Beacon Headers from block 17154001 to block 17169723 | 9024 / 15723 | queue
```

6698 | current 2694.81bps | total
> 3882943.63bps\
> Downloaded 17154031 / 17172359 | current 0.00bps | total 0.00bps
>
> Downloaded 17154062 / 17169724 | current 0.00bps | total 2.88bps

Keep in mind that resyncing a Nethermind node can take a considerable amount of time. It depends on your hardware,
internet connection, and the size of the blockchain.

consensus-clients.md:

---
title: Consensus clients
sidebarposition: 3

Ethereum's long-awaited shift from proof-of-work (PoW) to proof-of-stake (PoS) known as The Merge happened on September 15, 2022, and came with fundamental changes to the network. The most notable change is the addition of the consensus layer (aka Beacon Chain) which replaced the PoW mining. It is coordinating and pseudorandomly selecting block producers from the pool of stakers/validators in a way that makes it extremely difficult for validators to coordinate attacks on the network.
The Merge changed how operators run nodes on the Ethereum blockchain. A node now needs two clients that work together as a pair. In addition to the execution client (e.g., Nethermind), you need a consensus client that connects to the consensus layer and runs the PoS algorithm. This guide shows how to run an Ethereum node with Nethermind and a consensus client of your choice.

:::tip
An easy way to run both consensus and execution clients is with Sedge. Sedge is a setup tool for PoS validators and nodes that runs on Linux and macOS.
:::

Choosing a consensus client

On the consensus layer, there are 5 client implementations to choose from. Though all consensus clients are great, check them out yourself to find the one best suited to your needs.

- [Grandine][grandine]
- [Lighthouse][lighthouse]
- [Lodestar][lodestar]
- [Nimbus][nimbus]
- [Prysm][prysm]
- [Teku][teku]

:::warning Important
We urge you to take client diversity into consideration when choosing your consensus client and avoid the majority clients.
:::

Configuring JSON-RPC interface

Execution and consensus clients communicate via an authenticated endpoint specified in Engine JSON-RPC API. In order to connect to a consensus client, the execution client must generate a JWT secret at a known path. Although the secret is generated automatically by Nethermind on startup at keystore/jwt-secret path in its root directory, in some cases, you might need to do it yourself. You can generate one using OpenSSL:

bash

```
openssl rand -hex 32 > path/to/jwt.hex
```

:::note
Since the JWT secret is simply a 64-character hex value, there are many other ways of generating it, including online resources. However, for security reasons, we recommend using OpenSSL.
:::

The generated JWT secret can be specified with the --JsonRpc.JwtSecretFile path/to/jwt.hex command line option. For more configuration options, see Engine API.

Running the consensus client

This step assumes that you have already installed Nethermind, the consensus client of your choice, and, optionally, created the JWT secret.

:::info
As syncing from the scratch can take a very long time on some networks (up to several days), the commands below optionally use checkpoint sync to speed up the process.
:::

Grandine

```
bash
grandine \
  --network mainnet \
  --eth1-rpc-urls http://localhost:8551 \
  --jwt-secret path/to/jwt.hex \
  --checkpoint-sync-url https://beaconstate.ethstaker.cc
```

The command above runs Grandine on Mainnet. For other networks, set the --network and --checkpoint-sync-url options accordingly. See the [Grandine documentation][grandine] and [public checkpoint sync endpoints][checkpoint-sync-endpoints].

Lighthouse

```
bash
lighthouse bn \
  --network mainnet \
  --execution-endpoint http://localhost:8551 \
  --execution-jwt path/to/jwt.hex \
  --checkpoint-sync-url https://mainnet.checkpoint.sigp.io \
  --http
```

The command above runs Lighthouse on Mainnet. For other networks, set the --network and --checkpoint-sync-url options accordingly. See the [Lighthouse documentation][lighthouse] and [public checkpoint sync endpoints][checkpoint-sync-endpoints].

Lodestar

```
bash
lodestar beacon \
  --network mainnet \
  --jwt-secret path/to/jwt.hex \
  --checkpointSyncUrl https://beaconstate-mainnet.chainsafe.io
```

The command above runs Lodestar on Mainnet. For other networks, set the --

network and --checkpointSyncUrl options accordingly. See the [Lodestar documentation][lodestar] and [public checkpoint sync endpoints][checkpoint-sync-endpoints].

Nimbus

```
bash
./run-mainnet-beacon-node.sh \
  --web3-url=http://127.0.0.1:8551 \
  --jwt-secret=path/to/jwt.hex
```

The command above runs Numbus on Mainnet without checkpoint sync. For checkpoint sync, see Sync from a trusted node. For other networks, see the [Nimbus documentation][nimbus].

Prysm

```
bash
./prysm.sh beacon-chain \
  --mainnet \
  --execution-endpoint=http://localhost:8551 \
  --jwt-secret=path/to/jwt.hex \
  --checkpoint-sync-url=https://beaconstate.ethstaker.cc \
  --genesis-beacon-api-url=https://beaconstate.ethstaker.cc
```

The command above runs Prysm on Mainnet. For other networks, replace the --mainnet and set --checkpoint-sync-url and --genesis-beacon-api-url options accordingly. See the [Prysm documentation][prysm] and [public checkpoint sync endpoints][checkpoint-sync-endpoints].

Teku

```
bash
teku \
  --network=mainnet \
  --ee-endpoint=http://localhost:8551 \
  --ee-jwt-secret-file=path/to/jwt.hex \
  --metrics-enabled=true \
  --rest-api-enabled=true \
  --initial-state=https://beaconstate.ethstaker.cc
```

The command above runs Teku on Mainnet. For other networks, set the --network and --initial-state options accordingly. See the [Teku documentation][teku] and [public checkpoint sync endpoints][checkpoint-sync-endpoints].

Running Nethermind

```
:::warning Important
The consensus client must be running before you start Nethermind.
:::
```

```
bash
nethermind \
  -c mainnet \
  --JsonRpc.JwtSecretFile path/to/jwt.hex
```

The command above runs Nethermind on Mainnet. For other networks, set the -c option accordingly. For more info, see Running Nethermind.

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

```
endpoints
[grandine]: https://docs.grandine.io
[lighthouse]: https://lighthouse-book.sigmaprime.io
[lodestar]: https://chainsafe.github.io/lodestar
[nimbus]: https://nimbus.guide
[prysm]: https://docs.prylabs.network
[teku]: https://docs.teku.consensys.net

# installing-nethermind.md:

---
title: Installing Nethermind
sidebarposition: 2
---

import Tabs from "@theme/Tabs";
import TabItem from "@theme/TabItem";

Nethermind can be installed in several ways:

- Via a package manager
- As a standalone download
- As a Docker container
- By building from source code

Prerequisites

:::info
Does not apply to Docker distributions.
:::

Before installing Nethermind, your specific platform might need the following
prerequisites.

<Tabs groupId="os">
<TabItem value="linux" label="Linux">None</TabItem>
<TabItem value="windows" label="Windows">

Although the modern versions of Windows are bundled with a recent version of
Microsoft Visual C++ Redistributable, in some cases, it may need an update:

powershell
winget install Microsoft.VCRedist.2015+.x64


</TabItem>
<TabItem value="macos" label="macOS">None</TabItem>
</Tabs>

Package managers

Package managers are the easiest and fastest way of installing Nethermind.

:::tip
If you're using a package manager, it's highly recommended to set the -dd, --
datadir flag to specify the data directory. Otherwise, Nethermind will use the
default data directory where the package is installed, which may not be
preserved on further updates or uninstall.
:::

<Tabs groupId="os">
<TabItem value="linux" label="Linux">

On Ubuntu and other Linux distros supporting PPA, Nethermind can be installed
```

via Launchpad PPA.

First, add the Nethermind repository:

bash
sudo add-apt-repository ppa:nethermindeth/nethermind

:::note
If the command is not found, run:

bash
sudo apt-get install software-properties-common

:::

Then, install Nethermind as follows:

bash
sudo apt-get update
sudo apt-get install nethermind

</TabItem>
<TabItem value="windows" label="Windows">

On Windows, Nethermind can be installed via Windows Package Manager as follows:

powershell
winget install nethermind

</TabItem>
<TabItem value="macos" label="macOS">

On macOS, Nethermind can be installed via Homebrew.

First, add the Nethermind repository:

sh
brew tap nethermindeth/nethermind

Then, install Nethermind as follows:

sh
brew install nethermind

</TabItem>
</Tabs>

For further instructions, see Running Nethermind.

Standalone downloads

Standalone downloads give users more flexibility by allowing them to install a specific version of Nethermind, choose the installation location, and prevent automatic updates.

Standalone downloads are available on GitHub Releases and at downloads.nethermind.io as ZIP archives for x64 and AArch64 (ARM64) CPU architectures for Linux, Windows, and macOS.

Configuring as a Linux service

To install Nethermind as a Linux service, see the nethermind.service unit file as an example.
As it's configured to run Nethermind as the specific user and group and looks for the executable in a predefined location, the following steps need to be fulfilled:

:::note
Any of these steps can be omitted by replacing them with corresponding changes in the unit file.
For instance, if you want to run Nethermind as a different user, change the User and Group options in the unit file.
:::

1. Create a new user and group:

   bash
     Create a new user and group
     sudo useradd -m -s /bin/bash nethermind

     Increase the maximum number of open files
     sudo bash -c 'echo "nethermind soft nofile 100000" >
/etc/security/limits.d/nethermind.conf'
     sudo bash -c 'echo "nethermind hard nofile 100000" >>
/etc/security/limits.d/nethermind.conf'

     Switch to the nethermind user
     sudo su -l nethermind

     Create required directories
     Note that the home directory () is now /home/nethermind
     mkdir /build
     mkdir /data

2. Download Nethermind and extract the package contents to the /build directory.
3. Configure options in the /.env file:

   bash title="/.env"
   Required
   NETHERMINDCONFIG="mainnet"

   Optional
   NETHERMINDHEALTHCHECKSCONFIGENABLED="true"

Now, let's set up the Linux service:

bash
Download the unit file
curl -L https://raw.githubusercontent.com/NethermindEth/nethermind/master/
scripts/nethermind.service -o nethermind.service

Move the unit file to the systemd directory
sudo mv nethermind.service /etc/systemd/system

Reload the systemd daemon
sudo systemctl daemon-reload

Start the service
sudo systemctl start nethermind

Optionally, enable the service to start on boot
sudo systemctl enable nethermind

To ensure the service is up and running, check its status as follows:

bash
sudo systemctl status nethermind

To monitor the Nethermind output, run:

bash
journalctl -u nethermind -f

For further instructions, see Running Nethermind.

Docker container

The Docker images of Nethermind are available on Docker Hub.

This registry provides production versions of Nethermind with 3 types of tags:

- nethermind/nethermind:latest is the latest version of Nethermind (the default tag)
- nethermind/nethermind:<version> is the specific version of Nethermind where <version> is the actual version of Nethermind.
- nethermind/nethermind:<version>-chiseled is a rootless and chiseled image with the specific version of Nethermind where <version> is either latest or the actual version of Nethermind.\
  For security reasons, this image contains only the absolutely necessary components and is intended to run as a non-root app user with UID/GID of 64198.

To download the image from the registry, run:

bash
docker pull nethermind/nethermind

Starting a node is achieved by:

bash
docker run -it nethermind/nethermind

The following ports are exposed by default:

- 8545: TCP, for the JSON-RPC interface
- 8551: TCP, for the consensus client JSON-RPC interface
- 30303: TCP and UDP, for P2P networking

:::tip
It's highly recommended to mount data volumes as the Nethermind's data directories to ensure the synced data is preserved between the container restarts.
:::

The following volume mount points are available by default:

- /nethermind/netherminddb: used to store the database
- /nethermind/logs: used to store the logs
- /nethermind/keystore: used to store the keys

To mount separate volumes for each directory listed above, run:

```
bash
docker run -it \
  --mount type=bind,source=path/to/db,target=/nethermind/netherminddb \
  --mount type=bind,source=path/to/logs,target=/nethermind/logs \
  --mount type=bind,source=path/to/keystore,target=/nethermind/keystore \
  nethermind/nethermind
```

Alternatively, a single volume can be specified as the Nethermind data directory as follows:

```
bash
docker run -it \
  --mount type=bind,source=path/to/datadir,target=/nethermind/datadir \
  nethermind/nethermind -dd /nethermind/datadir
```

Note that any Nethermind-specific configuration option can be specified at the end. For instance, the -dd option in this case. For further instructions, see Running Nethermind.

To build the Docker image yourself, see Building Docker image.

Running Nethermind {#running}

:::warning Important

- A consensus client of your choice must be running before you start Nethermind.
- Please check out the security considerations before using Nethermind for critical operations.
  :::

Nethermind is mainly controlled by command line options (aka arguments or flags).

The full list of options can be displayed by running:

```
bash
nethermind -h
```

For instance, to launch the client with the default configuration for the Mainnet and custom data directory, run:

```
bash
nethermind -c mainnet -dd path/to/data/dir
```

For detailed info about the available configuration options, see Configuration.

Supported networks

To run Nethermind on a specific network, use the -c, --config command line option. Currently, the following networks are supported out of the box:

- Ethereum
  - Mainnet
  - Holesky (testnet)
  - Sepolia (testnet)
- Base
  - Base Mainnet
  - Base Sepolia (testnet)

- Energy Web Chain
  - Energy Web
  - Volta (testnet)
- Gnosis Chain
  - Gnosis
  - Chiado (testnet)
- Optimism
  - OP Mainnet
  - OP Sepolia (testnet)

intro.md:

---
title: Introduction and overview
sidebarlabel: Introduction
sidebarposition: 0
slug: /

Nethermind is a high-performance, highly configurable Ethereum execution client built on .NET that runs on Linux, Windows, and macOS and supports Clique, Aura, and Ethash. With breakneck sync speeds and support for external plugins, it provides reliable access to rich on-chain data thanks to a high-performance JSON-RPC interface and node health monitoring with Grafana and Seq.

Founded in 2017 and boosted by a grant from the Ethereum Foundation in 2018, we focus on delivering a robust Ethereum client, ensuring outstanding performance and flexibility for node operators.

Here, you'll find instructions on installing, configuring, and using Nethermind and its features. We've organized the information by topic and included examples and visuals for better understanding.

We hope you find this documentation helpful and welcome your feedback and suggestions!

migrating-from-geth.md:

---
title: Migrating from Geth
sidebarposition: 4

This guide will walk you through all the steps required for a seamless and quick transition.

First, ensure the disk has enough space. The most secure way is having a Nethermind synced on the same machine as Geth without shutting Geth down. That allows you to check whether Nethermind is syncing properly, verify whether everything works as expected, and reduce node downtime to a bare minimum. This is a recommended approach for any public JSON-RPC provider or a validator.

- If the disk has enough space, option 1 or option 2 are the recommended choices.
- If there is not enough disk space, and downtime is not an option, we recommend either extending the disk or, if not possible, hosting Nethermind on another machine, syncing it, and whenever it completes, moving everything validator-related to that machine, and abandoning the first one.
- If downtime of around 12 hours or more is not a problem, see the option 3.

Option 1: Sync Nethermind next to Geth {#option-1}

First, install Nethermind and a consensus client of your choice.

:::warning
- If you choose the same consensus client for Nethermind that is already being used with Geth, ensure their settings, such as data directories, do not interfere.
- Ensure the network ports of the consensus client paired with Nethermind and the one paired with Geth do not interfere with each other.
- Ensure the JSON-RPC port, Engine API port, and the P2P networking ports of Nethermind are different from the ones used by Geth. These ports are set using the following command line options:
  - --JsonRpc.Port <port>
  - --JsonRpc.EnginePort <port>
  - --Network.DiscoveryPort <port>
  - --Network.P2PPort <port>
:::

Once you fulfill the above requirements, you can start syncing Nethermind. To check the sync status, use the ethsyncing JSON-RPC method. When it returns false, Nethermind is considered fully synced with all block bodies and receipts needed to work properly as a validator. Another option to monitor the sync is a health check.

Once Nethermind is synced, shut down both Geth and Nethermind, along with its paired consensus client. Then, restart Nethermind using the ports assigned for and the JWT secret used by Geth before. Ensure no warnings or errors are present in the logs of both Nethermind and the consensus client. Also, check if Nethermind is following the chain properly. If everything is alright, you can remove Geth and the consensus client paired with Nethermind previously along with their data.

Option 2: Sync Nethermind using Sedge next to Geth {#option-2}

Sedge is a setup tool for PoS validators and nodes that runs on Linux and macOS.

This option is similar to the option 1, but Sedge automatically takes care of conflicting settings making the entire process much easier.

:::tip
You can add a flag to Sedge as follows. For instance:

- For the execution client, --el-extra-flag JsonRpc.Port=8546
- For the consensus client, --cl-extra-flag rpc-port=4001
:::

Once Nethermind is synced, you can remove extra flags if any from the docker-compose.yml and restart the node as follows:

bash
docker compose stop execution
docker compose up -d execution

Option 3: Remove Geth and sync Nethermind {#option-3}

This is the simplest option as it doesn't require configuration adjustments. However, the node will be down until Nethermind is syncing.

- Shut down and remove Geth along with its data.
- Install Nethermind
- Ensure Nethermind uses the same network ports as Geth before and the same JWT secret. Otherwise, you must reconfigure the consensus client to the Nethermind settings.

Once you fulfill the above requirements, you can start syncing Nethermind. While Nethermid is syncing, ensure no errors are present in the logs of both Nethermind and the consensus client. Note that sync may take a while, depending on the chain. Also, you can periodically check the ethsyncing JSON-RPC method or the health check.

system-requirements.md:

---
title: System requirements
sidebarposition: 1

Supported operating systems

Nethermind supports a broad range of modern 64-bit operating systems including but not limited to:

- Linux
    - Alpine 3.17+
    - CentOS Stream 9+
    - Debian 11+
    - Fedora 37+
    - openSUSE 15+
    - RHEL 8+
    - SLES 15+
    - Ubuntu 20.04+
- Windows
    - Windows 10+ (x64 only)
    - Windows Server 2012+ (x64 only)
- macOS 10.15+

Hardware requirements

Suggested requirements can be found below.

Network	Memory	CPU cores
Mainnet	16 GB	4
Mainnet archive	128 GB	8
Gnosis	16 GB	2
Energy Web	8 GB	2
Volta	8 GB	2

Disk requirements

Running an Ethereum Mainnet full node requires at least a 1 TB fast disk, such as NVMe or a fast SSD. However, 2
TB is recommended to minimize maintenance requirements. To choose a specific disk model, we recommend
checking out this guide.\
Choosing a 2 TB disk is a comfortable option for most common Mainnet node usage patterns, including staking.
Nonetheless, for a comprehensive understanding of disk growth and usage, we recommend delving further into the topic.

Database growth

Nethermind requires approximately 898 GB of disk space (as of Jan 2024) after a fresh Ethereum Mainnet sync using default parameters. This size increases over time as the Ethereum chain grows. The node's database is in its most optimal state immediately after a sync or full pruning. Following the initial sync, the database grows at a rate of approximately 27 GB per week. To maintain this

process, occasional resyncing or pruning of the node is necessary to bring it back to its optimal database size. For more info on managing node disk usage growth, see how to reduce database size.\
Having a larger disk space allocation reduces the need for frequent maintenance and alleviates concerns about the chain outgrowing the available space. It's worth noting that the only drawback of running a smaller disk is the requirement for more regular resyncing or pruning. We believe that a 2 TB disk will suffice for most users. However, the choice between 1 TB and 2 TB depends on factors such as hardware costs, cloud provider expenses, and individual requirements.

For more details, see Database size.

Disk speed

The speed of the disk often acts as a bottleneck for the node's performance. It is crucial for optimizing your validator's performance rewards and the syncing process. We highly recommend a disk with a minimum of 10,000 IOPS for both write and read operations. Slower disks may hinder your ability to synchronize the blockchain successfully.

Sync modes and disk usage

A node can be run using different syncing options, and each option has different characteristics of disk space usage.

- Archive node. This mode stores the full historical state for all blocks. As of July 2023, an archive node requires at least 14 TB of disk space, and it grows by approximately 60 GB per week.
- Ancient barriers. Nethermind allows you to specify how many old block bodies and receipts you want to store. By default, Nethermind sets the ancient barrier at block 11052984. This block is significant because it marks the deployment of the deposit contract required for consensus client deposit scanning for validators. Peers and JSON-RPC requests will not have access to block bodies and receipts older than the ancient barrier.
- Non-validator mode. This mode drops all historical receipts and bodies, but it cannot be used for validation.

For more details, see sync modes.

json-rpc-server.md:

---
title: JSON-RPC server
sidebarposition: 0

Interacting with Nethermind requires using the JSON-RPC 2.0 protocol. Nethermind provides JSON-RPC over HTTP, WebSocket, and IPC socket transports. Each transport must be enabled with the respective configuration option, as shown below. For more details, see the JSON-RPC configuration options.

The JSON-RPC API methods are grouped into several categories (namespaces) depending on their purpose. All API method names are composed of the namespace and the actual method name in that namespace. For instance, the ethcall method belongs to the eth namespace. See the sidebar for all supported namespaces and methods.

:::note
Not all of the JSON-RPC namespaces are enabled by default. Instead, they must be enabled explicitly with the --JsonRpc.EnabledModules command line option. Otherwise, error code -32600 is returned. The enabled namespaces can be found in the configuration file specified with the --config command line option.
:::

Transports

:::tip
The right choice of transport depends on the specific use case.

- HTTP is a familiar and idempotent transport that closes connections between requests and can, therefore, have lower overall overhead for a relatively low number of requests.
- WebSocket provides a continuous open channel that enables event subscriptions and streaming and handles large volumes of requests with more negligible per-message overhead.
- IPC is generally the most secure as it is limited to local interactions and cannot be exposed to external traffic. It can also be used for event subscriptions.
:::

HTTP

HTTP is the most widely used transport for Nethermind. To enable the HTTP server, use the --JsonRpc.Enabled true command line option. By default, Nethermind uses local loopback (127.0.0.1 or localhost) and 8545 port. To use a different host or port, set the --JsonRpc.Host and --JsonRpc.Port command line options, respectively.

WebSocket

The configuration of the WebSocket server follows the same pattern as the HTTP server. The WebSocket server is enabled automatically when the HTTP server is enabled and uses the same host and port. To disable the WebSocket server, use the --Init.WebSocketsEnabled false command line option. To use a different a port, set the --JsonRpc.WebSocketsPort command line option.

IPC socket

Nethermind uses IPC based on Unix domain socket. To enable the IPC server, use the --JsonRpc.IpcUnixDomainSocketPath path/to/ipc command line option. If the path/to/ipc doesn't exist, Nethermind creates one.

Engine API

The Engine API is a set of RPC methods that enable communication between an execution and consensus client. The clients call these methods automatically when they need to exchange information. Engine API is enabled automatically by default and is not designed to be exposed to the user.

By default, the Engine API uses local loopback (127.0.0.1 or localhost) and 8551 port. To use a different host or port, set the --JsonRpc.EngineHost and --JsonRpc.EnginePort command line options, respectively. For example, this can be useful when execution and consensus clients are on different machines.

:::warning Important
When the --JsonRpc.EngineHost option is specified, the --JsonRpc.EnginePort option must be specified as well.
:::

The Engine API uses JWT authentication and requires a JWT secret. By default, Nethermind creates one at keystore/jwt-secret path in its root directory. To use a different path, specify the --JsonRpc.JwtSecretFile path/to/jwt.hex command line option.

admin.md:

```
title: admin namespace
sidebarlabel: admin
sidebarposition: 0
---

import Tabs from "@theme/Tabs";
import TabItem from "@theme/TabItem";

adminaddPeer

Adds given node.

<Tabs>
<TabItem value="params" label="Parameters">

1. enode: string

2. addToStaticNodes: boolean


</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "adminaddPeer",
      "params": [enode, addToStaticNodes]
    }'


</TabItem>
<TabItem value="response" label="Response">

Added node

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: string

</TabItem>
</Tabs>

adminnodeInfo

Displays relevant information about this node.

<Tabs>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
```

```
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "adminnodeInfo",
      "params": []
    }'


</TabItem>
<TabItem value="response" label="Response">

Information about this node

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: object
  - enode: string
  - id: string
  - ip: string
  - listenAddress: string
  - name: string
  - ports: object
    - discovery: string (hex integer)
    - listener: string (hex integer)
  - protocols: map of object
    - difficulty: string (hex integer)
    - genesisHash: string (hash)
    - headHash: string (hash)
    - newtorkId: string (hex integer)

</TabItem>
</Tabs>

adminpeers

Displays a list of connected peers including information about them (clientId,
host, port, address, isBootnode, isStatic, enode).

<Tabs>
<TabItem value="params" label="Parameters">

1. includeDetails: boolean


</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "adminpeers",
      "params": [includeDetails]
    }'
```

```
</TabItem>
<TabItem value="response" label="Response">

List of connected peers including information

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: array of object
  - address: string
  - clientId: string
  - clientType: string
  - enode: string
  - ethDetails: string
  - host: string
  - isBootnode: boolean
  - isStatic: boolean
  - isTrusted: boolean
  - lastSignal: string
  - port: string (hex integer)

</TabItem>
</Tabs>

adminprune

Runs full pruning if enabled.

<Tabs>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "adminprune",
      "params": []
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: integer

</TabItem>
</Tabs>
```

adminremovePeer

Removes given node.

<Tabs>
<TabItem value="params" label="Parameters">

1. enode: string

2. removeFromStaticNodes: boolean

</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "adminremovePeer",
      "params": [enode, removeFromStaticNodes]
    }'

</TabItem>
<TabItem value="response" label="Response">

Removed node

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}

result: string

</TabItem>
</Tabs>

clique.md:

---
title: clique namespace
sidebarlabel: clique
sidebarposition: 1
---

import Tabs from "@theme/Tabs";
import TabItem from "@theme/TabItem";

cliquediscard

This method drops a currently running proposal. The signer will not cast further votes (either for or against) the address.

<Tabs>
<TabItem value="params" label="Parameters">

1. signer: string (address)

</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "cliquediscard",
      "params": [signer]
    }'

</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}

result: boolean

</TabItem>
</Tabs>

cliquegetBlockSigner

Retrieves the signer of the block with the given hash. Returns error of a block with the given hash does not exist.

<Tabs>
<TabItem value="params" label="Parameters">

1. hash: string (hash)

</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "cliquegetBlockSigner",
      "params": [hash]
    }'

</TabItem>

<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}

result: string (address)

</TabItem>
</Tabs>

cliquegetSigners

Retrieves the list of authorized signers.

<Tabs>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "cliquegetSigners",
      "params": []
    }'

</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}

result: array of string (address)

</TabItem>
</Tabs>

cliquegetSignersAnnotated

Retrieves the list of authorized signers but with signer names instead of addresses

<Tabs>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",

```
      "id": 0,
      "method": "cliquegetSignersAnnotated",
      "params": []
    }'
```

</TabItem>
<TabItem value="response" label="Response">

```
json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}
```

result: array of string

</TabItem>
</Tabs>

cliquegetSignersAtHash

Retrieves the list of authorized signers at the specified block by hash.

<Tabs>
<TabItem value="params" label="Parameters">

1. hash: string (hash)

</TabItem>
<TabItem value="request" label="Request" default>

```
bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "cliquegetSignersAtHash",
      "params": [hash]
    }'
```

</TabItem>
<TabItem value="response" label="Response">

```
json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}
```

result: array of string (address)

</TabItem>
</Tabs>

cliquegetSignersAtHashAnnotated

Retrieves the list of authorized signers at the specified block by hash but with signer names instead of addresses

<Tabs>
<TabItem value="params" label="Parameters">

1. hash: string (hash)

</TabItem>
<TabItem value="request" label="Request" default>

bash
```
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "cliquegetSignersAtHashAnnotated",
      "params": [hash]
    }'
```


</TabItem>
<TabItem value="response" label="Response">

json
```
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}
```


result: array of string

</TabItem>
</Tabs>

cliquegetSignersAtNumber

Retrieves the list of authorized signers at the specified block by block number.

<Tabs>
<TabItem value="params" label="Parameters">

1. number: string (hex integer)

</TabItem>
<TabItem value="request" label="Request" default>

bash
```
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "cliquegetSignersAtNumber",
      "params": [number]
    }'
```

```
</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: array of string (address)

</TabItem>
</Tabs>

cliquegetSnapshot

Retrieves a snapshot of all clique state at a given block.

<Tabs>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "cliquegetSnapshot",
      "params": []
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: object
  - hash: string (hash)
  - number: string (hex integer)
  - signerLimit: string (hex integer)
  - signers: map of string (hex integer)

</TabItem>
</Tabs>

cliquegetSnapshotAtHash

Retrieves the state snapshot at a given block.

<Tabs>
<TabItem value="params" label="Parameters">
```

1. hash: string (hash)

</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "cliquegetSnapshotAtHash",
      "params": [hash]
    }'

</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}

result: object
  - hash: string (hash)
  - number: string (hex integer)
  - signerLimit: string (hex integer)
  - signers: map of string (hex integer)

</TabItem>
</Tabs>

cliqueproduceBlock

Forces Clique block producer to produce a new block

<Tabs>
<TabItem value="params" label="Parameters">

1. parentHash: string (hash)

</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "cliqueproduceBlock",
      "params": [parentHash]
    }'

</TabItem>

<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}

result: boolean

</TabItem>
</Tabs>

cliquepropose

Adds a new authorization proposal that the signer will attempt to push through. If the vote parameter is true, the local signer votes for the given address to be included in the set of authorized signers. With vote set to false, the signer is against the address.

<Tabs>
<TabItem value="params" label="Parameters">

1. signer: string (address)

2. vote: boolean

</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "cliquepropose",
      "params": [signer, vote]
    }'

</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}

result: boolean

</TabItem>
</Tabs>

debug.md:

```
---
title: debug namespace
sidebarlabel: debug
sidebarposition: 2
---

import Tabs from "@theme/Tabs";
import TabItem from "@theme/TabItem";

debugdeleteChainSlice

Deletes a slice of a chain from the tree on all branches (Nethermind specific).

<Tabs>
<TabItem value="params" label="Parameters">

1. startNumber: string (hex integer)

2. force: boolean


</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "debugdeleteChainSlice",
      "params": [startNumber, force]
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: string (hex integer)

</TabItem>
</Tabs>

debuggetBadBlocks

Return list of invalid blocks.

<Tabs>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
```

```
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "debuggetBadBlocks",
      "params": []
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: array of object
  - block: object
    - author: string (address)
    - baseFeePerGas: string (hex integer)
    - blobGasUsed: string (hex integer)
    - difficulty: string (hex integer)
    - excessBlobGas: string (hex integer)
    - extraData: string (hex data)
    - gasLimit: string (hex integer)
    - gasUsed: string (hex integer)
    - hash: string (hash)
    - logsBloom: string (hex data)
    - miner: string (address)
    - mixHash: string (hash)
    - nonce: string (hex data)
    - number: string (hex integer)
    - parentBeaconBlockRoot: string (hash)
    - parentHash: string (hash)
    - receiptsRoot: string (hash)
    - sha3Uncles: string (hash)
    - signature: string (hex data)
    - size: string (hex integer)
    - stateRoot: string (hash)
    - step: string (hex integer)
    - timestamp: string (hex integer)
    - totalDifficulty: string (hex integer)
    - transactions: array of object
    - transactionsRoot: string (hash)
    - uncles: array of string (hash)
    - withdrawals: array of object
      - address: string (address)
      - amountInGwei: string (hex integer)
      - amountInWei: string (hex integer)
      - index: string (hex integer)
      - validatorIndex: string (hex integer)
    - withdrawalsRoot: string (hash)
  - hash: string (hash)
  - rlp: string (hex data)

</TabItem>
</Tabs>

debuggetBlockRlp

Retrieves a block in the RLP-serialized form.
```

<Tabs>
<TabItem value="params" label="Parameters">

1. number: string (hex integer)

</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "debuggetBlockRlp",
      "params": [number]
    }'

</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}

result: string (hex data)

</TabItem>
</Tabs>

debuggetBlockRlpByHash

Retrieves a block in the RLP-serialized form.

<Tabs>
<TabItem value="params" label="Parameters">

1. hash: string (hash)

</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "debuggetBlockRlpByHash",
      "params": [hash]
    }'

</TabItem>

<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}

result: string (hex data)

</TabItem>
</Tabs>

debuggetChainLevel

Retrieves a representation of tree branches on a given chain level (Nethermind specific).

<Tabs>
<TabItem value="params" label="Parameters">

1. number: string (hex integer)

</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "debuggetChainLevel",
      "params": [number]
    }'

</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}

result: object
  - blockInfos: array of object
    - blockHash: string (hash)
    - isFinalized: boolean
    - totalDifficulty: string (hex integer)
    - wasProcessed: boolean
  - hasBlockOnMainChain: boolean

</TabItem>
</Tabs>

debuggetConfigValue

Retrieves the Nethermind configuration value, e.g. JsonRpc.Enabled

<Tabs>
<TabItem value="params" label="Parameters">

1. category: string

2. name: string

</TabItem>
<TabItem value="request" label="Request" default>

bash
```
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "debuggetConfigValue",
      "params": [category, name]
    }'
```


</TabItem>
<TabItem value="response" label="Response">

json
```
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}
```


result: object

</TabItem>
</Tabs>

debuggetRawBlock

Get Raw Block format.

<Tabs>
<TabItem value="params" label="Parameters">

1. blockParameter: string (block number or hash or either of earliest, finalized, latest, pending, or safe)

</TabItem>
<TabItem value="request" label="Request" default>

bash
```
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "debuggetRawBlock",
```

```
      "params": [blockParameter]
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: string (hex data)

</TabItem>
</Tabs>

debuggetRawHeader

Get Raw Header format.

<Tabs>
<TabItem value="params" label="Parameters">

1. blockParameter: string (block number or hash or either of earliest,
finalized, latest, pending, or safe)


</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "debuggetRawHeader",
      "params": [blockParameter]
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: string (hex data)

</TabItem>
</Tabs>

debuggetRawReceipts
```

Get Raw Receipt format.

<Tabs>
<TabItem value="params" label="Parameters">

1. blockParameter: string (block number or hash or either of earliest, finalized, latest, pending, or safe)

</TabItem>
<TabItem value="request" label="Request" default>

bash
```
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "debuggetRawReceipts",
      "params": [blockParameter]
    }'
```

</TabItem>
<TabItem value="response" label="Response">

json
```
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}
```

result: array of string (hex data)

</TabItem>
</Tabs>

debuggetRawTransaction

Get Raw Transaction format.

<Tabs>
<TabItem value="params" label="Parameters">

1. transactionHash: string (hash)

</TabItem>
<TabItem value="request" label="Request" default>

bash
```
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "debuggetRawTransaction",
      "params": [transactionHash]
    }'
```

</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}

result: string (hex data)

</TabItem>
</Tabs>

debuggetSyncStage

Retrives Nethermind Sync Stage, With extra Metadata

<Tabs>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "debuggetSyncStage",
      "params": []
    }'

</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}

result: object
  - currentStage: string

</TabItem>
</Tabs>

debuginsertReceipts

Insert receipts for the block after verifying receipts root correctness.

<Tabs>
<TabItem value="params" label="Parameters">

1. blockParameter: string (block number or hash or either of earliest,
finalized, latest, pending, or safe)

2. receiptForRpc: array of object

```
  - blobGasPrice: string (hex integer)
  - blobGasUsed: string (hex integer)
  - blockHash: string (hash)
  - blockNumber: string (hex integer)
  - contractAddress: string (address)
  - cumulativeGasUsed: string (hex integer)
  - effectiveGasPrice: string (hex integer)
  - error: string
  - from: string (address)
  - gasUsed: string (hex integer)
  - logs: array of object
    - address: string (address)
    - blockHash: string (hash)
    - blockNumber: string (hex integer)
    - data: string (hex data)
    - logIndex: string (hex integer)
    - removed: boolean
    - topics: array of string (hash)
    - transactionHash: string (hash)
    - transactionIndex: string (hex integer)
  - logsBloom: string (hex data)
  - root: string (hash)
  - status: string (hex integer)
  - to: string (address)
  - transactionHash: string (hash)
  - transactionIndex: string (hex integer)
  - type: integer


</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "debuginsertReceipts",
      "params": [blockParameter, receiptForRpc]
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: boolean

</TabItem>
</Tabs>

debugmigrateReceipts

Sets the block number up to which receipts will be migrated to (Nethermind
specific).
```

<Tabs>
<TabItem value="params" label="Parameters">

1. blockNumber: string (hex integer)

</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "debugmigrateReceipts",
      "params": [blockNumber]
    }'

</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}

result: boolean

</TabItem>
</Tabs>

debugresetHead

Updates / resets head block - use only when the node got stuck due to DB / memory corruption (Nethermind specific).

<Tabs>
<TabItem value="params" label="Parameters">

1. blockHash: string (hash)

</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "debugresetHead",
      "params": [blockHash]
    }'

```
</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: boolean

</TabItem>
</Tabs>

debugstandardTraceBadBlockToFile

This method is similar to the debugstandardTraceBlockToFile method, but can be
used to obtain information about a block that has been rejected as invalid.

<Tabs>
<TabItem value="params" label="Parameters">

1. blockHash: string (hash)

2. options: object
    - disableMemory: boolean
    - disableStack: boolean
    - disableStorage: boolean
    - enableMemory: boolean
    - timeout: string
    - tracer: string
    - tracerConfig: object
      - hasValue: boolean
      - value: object
        - item: object
          <!--[circular ref]-->
        - valueKind: integer
    - txHash: string (hash)


</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "debugstandardTraceBadBlockToFile",
      "params": [blockHash, options]
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
```

```
  "result": result
}
```

result: array of string

</TabItem>
</Tabs>

debugstandardTraceBlockToFile

Writes to a file the full stack trace of all invoked opcodes of the transaction specified (or all transactions if not specified) that was included in the block specified. The parent of the block must be present or it will fail.

<Tabs>
<TabItem value="params" label="Parameters">

1. blockHash: string (hash)

2. options: object
    - disableMemory: boolean
    - disableStack: boolean
    - disableStorage: boolean
    - enableMemory: boolean
    - timeout: string
    - tracer: string
    - tracerConfig: object
      - hasValue: boolean
      - value: object
        - item: object
          <!--[circular ref]-->
        - valueKind: integer
    - txHash: string (hash)

</TabItem>
<TabItem value="request" label="Request" default>

bash
```
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "debugstandardTraceBlockToFile",
      "params": [blockHash, options]
    }'
```

</TabItem>
<TabItem value="response" label="Response">

json
```
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}
```

result: array of string

</TabItem>
</Tabs>

debugtraceBlock

Returns the full stack trace of all invoked opcodes of all transactions that were included in the block specified. The parent of the block must be present or it will fail.

<Tabs>
<TabItem value="params" label="Parameters">

1. blockRlp: string (hex data)

2. options: object
    - disableMemory: boolean
    - disableStack: boolean
    - disableStorage: boolean
    - enableMemory: boolean
    - timeout: string
    - tracer: string
    - tracerConfig: object
      - hasValue: boolean
      - value: object
        - item: object
          <!--[circular ref]-->
        - valueKind: integer
    - txHash: string (hash)

</TabItem>
<TabItem value="request" label="Request" default>

bash
```
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "debugtraceBlock",
      "params": [blockRlp, options]
    }'
```


</TabItem>
<TabItem value="response" label="Response">

json
```
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}
```


result: array of object
  - customTracerResult: object
    - value: object
  - entries: array of object
    - depth: string (hex integer)
    - error: string
    - gas: string (hex integer)
    - gasCost: string (hex integer)

```
    - memory: array of string
    - opcode: string
    - programCounter: string (hex integer)
    - stack: array of string
    - storage: map of string
  - failed: boolean
  - gas: string (hex integer)
  - returnValue: string (hex data)
  - storagesByDepth: array of map of string

</TabItem>
</Tabs>

debugtraceBlockByHash

Similar to debugtraceBlock, this method accepts a block hash and replays the
block that is already present in the database.

<Tabs>
<TabItem value="params" label="Parameters">

1. blockHash: string (hash)

2. options: object
    - disableMemory: boolean
    - disableStack: boolean
    - disableStorage: boolean
    - enableMemory: boolean
    - timeout: string
    - tracer: string
    - tracerConfig: object
      - hasValue: boolean
      - value: object
        - item: object
          <!--[circular ref]-->
        - valueKind: integer
    - txHash: string (hash)


</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "debugtraceBlockByHash",
      "params": [blockHash, options]
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}
```

```
result: array of object
  - customTracerResult: object
    - value: object
  - entries: array of object
    - depth: string (hex integer)
    - error: string
    - gas: string (hex integer)
    - gasCost: string (hex integer)
    - memory: array of string
    - opcode: string
    - programCounter: string (hex integer)
    - stack: array of string
    - storage: map of string
  - failed: boolean
  - gas: string (hex integer)
  - returnValue: string (hex data)
  - storagesByDepth: array of map of string

</TabItem>
</Tabs>

debugtraceBlockByNumber

Similar to debugtraceBlock, this method accepts a block number as well as
"latest" or "finalized" and replays the block that is already present in the
database.

<Tabs>
<TabItem value="params" label="Parameters">

1. blockParameter: string (block number or hash or either of earliest,
finalized, latest, pending, or safe)

2. options: object
    - disableMemory: boolean
    - disableStack: boolean
    - disableStorage: boolean
    - enableMemory: boolean
    - timeout: string
    - tracer: string
    - tracerConfig: object
      - hasValue: boolean
      - value: object
        - item: object
          <!--[circular ref]-->
        - valueKind: integer
    - txHash: string (hash)


</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "debugtraceBlockByNumber",
      "params": [blockParameter, options]
    }'
```

```
</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: array of object
  - customTracerResult: object
    - value: object
  - entries: array of object
    - depth: string (hex integer)
    - error: string
    - gas: string (hex integer)
    - gasCost: string (hex integer)
    - memory: array of string
    - opcode: string
    - programCounter: string (hex integer)
    - stack: array of string
    - storage: map of string
  - failed: boolean
  - gas: string (hex integer)
  - returnValue: string (hex data)
  - storagesByDepth: array of map of string

</TabItem>
</Tabs>

debugtraceCall

This method lets you run an ethcall within the context of the given block
execution using the final state of parent block as the base. The block can be
specified either by hash or by number. It takes the same input object as a
ethcall. It returns the same output as debugtraceTransaction.

<Tabs>
<TabItem value="params" label="Parameters">

1. call: object
    - accessList: array of object
      - address: string (address)
      - storageKeys: array of string (hex integer)
    - blobVersionedHashes: array of string (hex data)
    - blockHash: string (hash)
    - blockNumber: string (hex integer)
    - chainId: string (hex integer)
    - data: string (hex data)
    - from: string (address)
    - gas: string (hex integer)
    - gasPrice: string (hex integer)
    - hash: string (hash)
    - input: string (hex data)
    - isSystemTx: boolean
    - maxFeePerBlobGas: string (hex integer)
    - maxFeePerGas: string (hex integer)
    - maxPriorityFeePerGas: string (hex integer)
    - mint: string (hex integer)
    - nonce: string (hex integer)
    - r: string (hex integer)
```

```
    - s: string (hex integer)
    - sourceHash: string (hash)
    - to: string (address)
    - transactionIndex: string (hex integer)
    - type: integer
    - v: string (hex integer)
    - value: string (hex integer)
    - yParity: string (hex integer)

2. blockParameter: string (block number or hash or either of earliest,
finalized, latest, pending, or safe)

3. options: object
    - disableMemory: boolean
    - disableStack: boolean
    - disableStorage: boolean
    - enableMemory: boolean
    - timeout: string
    - tracer: string
    - tracerConfig: object
      - hasValue: boolean
      - value: object
        - item: object
          <!--[circular ref]-->
        - valueKind: integer
    - txHash: string (hash)


</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "debugtraceCall",
      "params": [call, blockParameter, options]
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: object
  - customTracerResult: object
    - value: object
  - entries: array of object
    - depth: string (hex integer)
    - error: string
    - gas: string (hex integer)
    - gasCost: string (hex integer)
    - memory: array of string
    - opcode: string
```

```
    - programCounter: string (hex integer)
    - stack: array of string
    - storage: map of string
  - failed: boolean
  - gas: string (hex integer)
  - returnValue: string (hex data)
  - storagesByDepth: array of map of string

</TabItem>
</Tabs>

debugtraceTransaction

This method will attempt to run the transaction in the exact same manner as it
was executed on the network. It will replay any transaction that may have been
executed prior to this one before it will finally attempt to execute the
transaction that corresponds to the given hash.

<Tabs>
<TabItem value="params" label="Parameters">

1. transactionHash: string (hash)

2. options: object
    - disableMemory: boolean
    - disableStack: boolean
    - disableStorage: boolean
    - enableMemory: boolean
    - timeout: string
    - tracer: string
    - tracerConfig: object
      - hasValue: boolean
      - value: object
        - item: object
          <!--[circular ref]-->
        - valueKind: integer
    - txHash: string (hash)


</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "debugtraceTransaction",
      "params": [transactionHash, options]
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}
```

```
result: object
  - customTracerResult: object
    - value: object
  - entries: array of object
    - depth: string (hex integer)
    - error: string
    - gas: string (hex integer)
    - gasCost: string (hex integer)
    - memory: array of string
    - opcode: string
    - programCounter: string (hex integer)
    - stack: array of string
    - storage: map of string
  - failed: boolean
  - gas: string (hex integer)
  - returnValue: string (hex data)
  - storagesByDepth: array of map of string

</TabItem>
</Tabs>

debugtraceTransactionByBlockAndIndex

<Tabs>
<TabItem value="params" label="Parameters">

1. blockParameter: string (block number or hash or either of earliest,
finalized, latest, pending, or safe)

2. txIndex: string (hex integer)

3. options: object
    - disableMemory: boolean
    - disableStack: boolean
    - disableStorage: boolean
    - enableMemory: boolean
    - timeout: string
    - tracer: string
    - tracerConfig: object
      - hasValue: boolean
      - value: object
        - item: object
          <!--[circular ref]-->
        - valueKind: integer
    - txHash: string (hash)


</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "debugtraceTransactionByBlockAndIndex",
      "params": [blockParameter, txIndex, options]
    }'


</TabItem>
```

<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}

result: object
  - customTracerResult: object
    - value: object
  - entries: array of object
    - depth: string (hex integer)
    - error: string
    - gas: string (hex integer)
    - gasCost: string (hex integer)
    - memory: array of string
    - opcode: string
    - programCounter: string (hex integer)
    - stack: array of string
    - storage: map of string
  - failed: boolean
  - gas: string (hex integer)
  - returnValue: string (hex data)
  - storagesByDepth: array of map of string

</TabItem>
</Tabs>

debugtraceTransactionByBlockhashAndIndex

<Tabs>
<TabItem value="params" label="Parameters">

1. blockHash: string (hash)

2. txIndex: string (hex integer)

3. options: object
    - disableMemory: boolean
    - disableStack: boolean
    - disableStorage: boolean
    - enableMemory: boolean
    - timeout: string
    - tracer: string
    - tracerConfig: object
      - hasValue: boolean
      - value: object
        - item: object
          <!--[circular ref]-->
        - valueKind: integer
    - txHash: string (hash)

</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{

```
      "jsonrpc": "2.0",
      "id": 0,
      "method": "debugtraceTransactionByBlockhashAndIndex",
      "params": [blockHash, txIndex, options]
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: object
  - customTracerResult: object
    - value: object
  - entries: array of object
    - depth: string (hex integer)
    - error: string
    - gas: string (hex integer)
    - gasCost: string (hex integer)
    - memory: array of string
    - opcode: string
    - programCounter: string (hex integer)
    - stack: array of string
    - storage: map of string
  - failed: boolean
  - gas: string (hex integer)
  - returnValue: string (hex data)
  - storagesByDepth: array of map of string

</TabItem>
</Tabs>

debugtraceTransactionInBlockByHash

<Tabs>
<TabItem value="params" label="Parameters">

1. blockRlp: string (hex data)

2. transactionHash: string (hash)

3. options: object
    - disableMemory: boolean
    - disableStack: boolean
    - disableStorage: boolean
    - enableMemory: boolean
    - timeout: string
    - tracer: string
    - tracerConfig: object
      - hasValue: boolean
      - value: object
        - item: object
          <!--[circular ref]-->
        - valueKind: integer
    - txHash: string (hash)
```

```
</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "debugtraceTransactionInBlockByHash",
      "params": [blockRlp, transactionHash, options]
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: object
  - customTracerResult: object
    - value: object
  - entries: array of object
    - depth: string (hex integer)
    - error: string
    - gas: string (hex integer)
    - gasCost: string (hex integer)
    - memory: array of string
    - opcode: string
    - programCounter: string (hex integer)
    - stack: array of string
    - storage: map of string
  - failed: boolean
  - gas: string (hex integer)
  - returnValue: string (hex data)
  - storagesByDepth: array of map of string

</TabItem>
</Tabs>

debugtraceTransactionInBlockByIndex

<Tabs>
<TabItem value="params" label="Parameters">

1. blockRlp: string (hex data)

2. txIndex: string (hex integer)

3. options: object
    - disableMemory: boolean
    - disableStack: boolean
    - disableStorage: boolean
    - enableMemory: boolean
    - timeout: string
    - tracer: string
    - tracerConfig: object
```

```
      - hasValue: boolean
      - value: object
        - item: object
          <!--[circular ref]-->
        - valueKind: integer
    - txHash: string (hash)


</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "debugtraceTransactionInBlockByIndex",
      "params": [blockRlp, txIndex, options]
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: object
  - customTracerResult: object
    - value: object
  - entries: array of object
    - depth: string (hex integer)
    - error: string
    - gas: string (hex integer)
    - gasCost: string (hex integer)
    - memory: array of string
    - opcode: string
    - programCounter: string (hex integer)
    - stack: array of string
    - storage: map of string
  - failed: boolean
  - gas: string (hex integer)
  - returnValue: string (hex data)
  - storagesByDepth: array of map of string

</TabItem>
</Tabs>



# eth.md:

---
title: eth namespace
sidebarlabel: eth
sidebarposition: 3
---
```

```
import Tabs from "@theme/Tabs";
import TabItem from "@theme/TabItem";

ethblobBaseFee

Returns the base fee per blob gas in wei

<Tabs>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "ethblobBaseFee",
      "params": []
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: string (hex integer)

</TabItem>
</Tabs>

ethblockNumber

Returns current block number

<Tabs>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "ethblockNumber",
      "params": []
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
```

```
  "id": 0,
  "result": result
}


result: string (hex integer)

</TabItem>
</Tabs>

ethcall

Executes a tx call (does not create a transaction)

<Tabs>
<TabItem value="params" label="Parameters">

1. transactionCall: object
    - accessList: array of object
      - address: string (address)
      - storageKeys: array of string (hex integer)
    - blobVersionedHashes: array of string (hex data)
    - blockHash: string (hash)
    - blockNumber: string (hex integer)
    - chainId: string (hex integer)
    - data: string (hex data)
    - from: string (address)
    - gas: string (hex integer)
    - gasPrice: string (hex integer)
    - hash: string (hash)
    - input: string (hex data)
    - isSystemTx: boolean
    - maxFeePerBlobGas: string (hex integer)
    - maxFeePerGas: string (hex integer)
    - maxPriorityFeePerGas: string (hex integer)
    - mint: string (hex integer)
    - nonce: string (hex integer)
    - r: string (hex integer)
    - s: string (hex integer)
    - sourceHash: string (hash)
    - to: string (address)
    - transactionIndex: string (hex integer)
    - type: integer
    - v: string (hex integer)
    - value: string (hex integer)
    - yParity: string (hex integer)

2. blockParameter: string (block number or hash or either of earliest,
finalized, latest, pending, or safe)


</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "ethcall",
      "params": [transactionCall, blockParameter]
    }'
```

```
</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: string

</TabItem>
</Tabs>

ethchainId

Returns ChainID

<Tabs>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "ethchainId",
      "params": []
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: string (hex integer)

</TabItem>
</Tabs>

ethcreateAccessList

Creates an EIP2930 type AccessList for the given transaction

<Tabs>
<TabItem value="params" label="Parameters">

1. transactionCall: object
    - accessList: array of object
      - address: string (address)
      - storageKeys: array of string (hex integer)
```

```
    - blobVersionedHashes: array of string (hex data)
    - blockHash: string (hash)
    - blockNumber: string (hex integer)
    - chainId: string (hex integer)
    - data: string (hex data)
    - from: string (address)
    - gas: string (hex integer)
    - gasPrice: string (hex integer)
    - hash: string (hash)
    - input: string (hex data)
    - isSystemTx: boolean
    - maxFeePerBlobGas: string (hex integer)
    - maxFeePerGas: string (hex integer)
    - maxPriorityFeePerGas: string (hex integer)
    - mint: string (hex integer)
    - nonce: string (hex integer)
    - r: string (hex integer)
    - s: string (hex integer)
    - sourceHash: string (hash)
    - to: string (address)
    - transactionIndex: string (hex integer)
    - type: integer
    - v: string (hex integer)
    - value: string (hex integer)
    - yParity: string (hex integer)

2. blockParameter: string (block number or hash or either of earliest,
finalized, latest, pending, or safe)

3. optimize: boolean


</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "ethcreateAccessList",
      "params": [transactionCall, blockParameter, optimize]
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: object
  - accessList: array of object
    - address: string (address)
    - storageKeys: array of string (hex integer)
  - gasUsed: string (hex integer)
```

</TabItem>
</Tabs>

ethestimateGas

Executes a tx call and returns gas used (does not create a transaction)

<Tabs>
<TabItem value="params" label="Parameters">

1. transactionCall: object
    - accessList: array of object
      - address: string (address)
      - storageKeys: array of string (hex integer)
    - blobVersionedHashes: array of string (hex data)
    - blockHash: string (hash)
    - blockNumber: string (hex integer)
    - chainId: string (hex integer)
    - data: string (hex data)
    - from: string (address)
    - gas: string (hex integer)
    - gasPrice: string (hex integer)
    - hash: string (hash)
    - input: string (hex data)
    - isSystemTx: boolean
    - maxFeePerBlobGas: string (hex integer)
    - maxFeePerGas: string (hex integer)
    - maxPriorityFeePerGas: string (hex integer)
    - mint: string (hex integer)
    - nonce: string (hex integer)
    - r: string (hex integer)
    - s: string (hex integer)
    - sourceHash: string (hash)
    - to: string (address)
    - transactionIndex: string (hex integer)
    - type: integer
    - v: string (hex integer)
    - value: string (hex integer)
    - yParity: string (hex integer)

2. blockParameter: string (block number or hash or either of earliest, finalized, latest, pending, or safe)

</TabItem>
<TabItem value="request" label="Request" default>

bash
```
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "ethestimateGas",
      "params": [transactionCall, blockParameter]
    }'
```

</TabItem>
<TabItem value="response" label="Response">

json
```
{
```

```
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: string (hex integer)

</TabItem>
</Tabs>

ethfeeHistory

Returns block fee history.

<Tabs>
<TabItem value="params" label="Parameters">

1. blockCount: string (hex integer)

2. newestBlock: string (block number or hash or either of earliest, finalized,
latest, pending, or safe)

3. rewardPercentiles: array of object


</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "ethfeeHistory",
      "params": [blockCount, newestBlock, rewardPercentiles]
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: object
  - baseFeePerBlobGas: array of string (hex integer)
  - baseFeePerGas: array of string (hex integer)
  - blobGasUsedRatio: array of object
  - gasUsedRatio: array of object
  - oldestBlock: string (hex integer)
  - reward: array of array of string (hex integer)

</TabItem>
</Tabs>

ethgasPrice
```

Returns miner's gas price

<Tabs>
<TabItem value="request" label="Request" default>

bash
```
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "ethgasPrice",
      "params": []
    }'
```

</TabItem>
<TabItem value="response" label="Response">

json
```
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}
```

result: string (hex integer)

</TabItem>
</Tabs>

ethgetAccount

Retrieves Accounts via Address and Blocknumber

<Tabs>
<TabItem value="params" label="Parameters">

1. accountAddress: string (address)

2. blockParameter: string (block number or hash or either of earliest, finalized, latest, pending, or safe)

</TabItem>
<TabItem value="request" label="Request" default>

bash
```
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "ethgetAccount",
      "params": [accountAddress, blockParameter]
    }'
```

</TabItem>
<TabItem value="response" label="Response">

```
json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: object
  - balance: string (hex integer)
  - codeHash: object
    - bytes: object
      - isEmpty: boolean
      - item: object
      - length: string (hex integer)
    - bytesAsSpan: object
      - isEmpty: boolean
      - item: object
      - length: string (hex integer)
  - nonce: string (hex integer)
  - storageRoot: object
    - bytes: object
      - isEmpty: boolean
      - item: object
      - length: string (hex integer)
    - bytesAsSpan: object
      - isEmpty: boolean
      - item: object
      - length: string (hex integer)

</TabItem>
</Tabs>

ethgetBalance

Returns account balance

<Tabs>
<TabItem value="params" label="Parameters">

1. address: string (address)

2. blockParameter: string (block number or hash or either of earliest,
finalized, latest, pending, or safe)


</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "ethgetBalance",
      "params": [address, blockParameter]
    }'


</TabItem>
<TabItem value="response" label="Response">
```

```
json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: string (hex integer)

</TabItem>
</Tabs>

ethgetBlockByHash

Retrieves a block by hash

<Tabs>
<TabItem value="params" label="Parameters">

1. blockHash: string (hash)

2. returnFullTransactionObjects: boolean


</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "ethgetBlockByHash",
      "params": [blockHash, returnFullTransactionObjects]
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: object
  - author: string (address)
  - baseFeePerGas: string (hex integer)
  - blobGasUsed: string (hex integer)
  - difficulty: string (hex integer)
  - excessBlobGas: string (hex integer)
  - extraData: string (hex data)
  - gasLimit: string (hex integer)
  - gasUsed: string (hex integer)
  - hash: string (hash)
  - logsBloom: string (hex data)
  - miner: string (address)
```

```
  - mixHash: string (hash)
  - nonce: string (hex data)
  - number: string (hex integer)
  - parentBeaconBlockRoot: string (hash)
  - parentHash: string (hash)
  - receiptsRoot: string (hash)
  - sha3Uncles: string (hash)
  - signature: string (hex data)
  - size: string (hex integer)
  - stateRoot: string (hash)
  - step: string (hex integer)
  - timestamp: string (hex integer)
  - totalDifficulty: string (hex integer)
  - transactions: array of object
  - transactionsRoot: string (hash)
  - uncles: array of string (hash)
  - withdrawals: array of object
    - address: string (address)
    - amountInGwei: string (hex integer)
    - amountInWei: string (hex integer)
    - index: string (hex integer)
    - validatorIndex: string (hex integer)
  - withdrawalsRoot: string (hash)

</TabItem>
</Tabs>

ethgetBlockByNumber

Retrieves a block by number

<Tabs>
<TabItem value="params" label="Parameters">

1. blockParameter: string (block number or hash or either of earliest,
finalized, latest, pending, or safe)

2. returnFullTransactionObjects: boolean


</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "ethgetBlockByNumber",
      "params": [blockParameter, returnFullTransactionObjects]
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}
```

```
result: object
  - author: string (address)
  - baseFeePerGas: string (hex integer)
  - blobGasUsed: string (hex integer)
  - difficulty: string (hex integer)
  - excessBlobGas: string (hex integer)
  - extraData: string (hex data)
  - gasLimit: string (hex integer)
  - gasUsed: string (hex integer)
  - hash: string (hash)
  - logsBloom: string (hex data)
  - miner: string (address)
  - mixHash: string (hash)
  - nonce: string (hex data)
  - number: string (hex integer)
  - parentBeaconBlockRoot: string (hash)
  - parentHash: string (hash)
  - receiptsRoot: string (hash)
  - sha3Uncles: string (hash)
  - signature: string (hex data)
  - size: string (hex integer)
  - stateRoot: string (hash)
  - step: string (hex integer)
  - timestamp: string (hex integer)
  - totalDifficulty: string (hex integer)
  - transactions: array of object
  - transactionsRoot: string (hash)
  - uncles: array of string (hash)
  - withdrawals: array of object
    - address: string (address)
    - amountInGwei: string (hex integer)
    - amountInWei: string (hex integer)
    - index: string (hex integer)
    - validatorIndex: string (hex integer)
  - withdrawalsRoot: string (hash)

</TabItem>
</Tabs>

ethgetBlockReceipts

Get receipts from all transactions from particular block, more efficient than
fetching the receipts one-by-one.

<Tabs>
<TabItem value="params" label="Parameters">

1. blockParameter: string (block number or hash or either of earliest,
finalized, latest, pending, or safe)


</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "ethgetBlockReceipts",
```

```
      "params": [blockParameter]
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: array of object
  - blobGasPrice: string (hex integer)
  - blobGasUsed: string (hex integer)
  - blockHash: string (hash)
  - blockNumber: string (hex integer)
  - contractAddress: string (address)
  - cumulativeGasUsed: string (hex integer)
  - effectiveGasPrice: string (hex integer)
  - error: string
  - from: string (address)
  - gasUsed: string (hex integer)
  - logs: array of object
    - address: string (address)
    - blockHash: string (hash)
    - blockNumber: string (hex integer)
    - data: string (hex data)
    - logIndex: string (hex integer)
    - removed: boolean
    - topics: array of string (hash)
    - transactionHash: string (hash)
    - transactionIndex: string (hex integer)
  - logsBloom: string (hex data)
  - root: string (hash)
  - status: string (hex integer)
  - to: string (address)
  - transactionHash: string (hash)
  - transactionIndex: string (hex integer)
  - type: integer

</TabItem>
</Tabs>

ethgetBlockTransactionCountByHash

Returns number of transactions in the block block hash

<Tabs>
<TabItem value="params" label="Parameters">

1. blockHash: string (hash)


</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
```

```
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "ethgetBlockTransactionCountByHash",
      "params": [blockHash]
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: string (hex integer)

</TabItem>
</Tabs>

ethgetBlockTransactionCountByNumber

Returns number of transactions in the block by block number

<Tabs>
<TabItem value="params" label="Parameters">

1. blockParameter: string (block number or hash or either of earliest,
finalized, latest, pending, or safe)


</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "ethgetBlockTransactionCountByNumber",
      "params": [blockParameter]
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: string (hex integer)

</TabItem>
```

```
</Tabs>

ethgetCode

Returns account code at given address and block

<Tabs>
<TabItem value="params" label="Parameters">

1. address: string (address)

2. blockParameter: string (block number or hash or either of earliest,
finalized, latest, pending, or safe)


</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "ethgetCode",
      "params": [address, blockParameter]
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: string (hex data)

</TabItem>
</Tabs>

ethgetFilterChanges

Reads filter changes

<Tabs>
<TabItem value="params" label="Parameters">

1. filterId: string (hex integer)


</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
```

```
      "jsonrpc": "2.0",
      "id": 0,
      "method": "ethgetFilterChanges",
      "params": [filterId]
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: array of object

</TabItem>
</Tabs>

ethgetFilterLogs

Reads filter changes

<Tabs>
<TabItem value="params" label="Parameters">

1. filterId: string (hex integer)


</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "ethgetFilterLogs",
      "params": [filterId]
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: array of object
  - address: string (address)
  - blockHash: string (hash)
  - blockNumber: string (hex integer)
  - data: string (hex data)
```

```
  - logIndex: string (hex integer)
  - removed: boolean
  - topics: array of string (hash)
  - transactionHash: string (hash)
  - transactionIndex: string (hex integer)
  - transactionLogIndex: string (hex integer)

</TabItem>
</Tabs>

ethgetLogs

Reads logs

<Tabs>
<TabItem value="params" label="Parameters">

1. filter: object
    - address: object
    - fromBlock: string (block number or hash or either of earliest, finalized,
latest, pending, or safe)
    - toBlock: string (block number or hash or either of earliest, finalized,
latest, pending, or safe)
    - topics: array of object


</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "ethgetLogs",
      "params": [filter]
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: array of object
  - address: string (address)
  - blockHash: string (hash)
  - blockNumber: string (hex integer)
  - data: string (hex data)
  - logIndex: string (hex integer)
  - removed: boolean
  - topics: array of string (hash)
  - transactionHash: string (hash)
  - transactionIndex: string (hex integer)
  - transactionLogIndex: string (hex integer)
```

```
</TabItem>
</Tabs>

ethgetProof

https://github.com/ethereum/EIPs/issues/1186

<Tabs>
<TabItem value="params" label="Parameters">

1. accountAddress: string (address)

2. hashRate: array of string (hex integer)

3. blockParameter: string (block number or hash or either of earliest,
finalized, latest, pending, or safe)


</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "ethgetProof",
      "params": [accountAddress, hashRate, blockParameter]
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: object
  - address: string (address)
  - balance: string (hex integer)
  - codeHash: string (hash)
  - nonce: string (hex integer)
  - proof: array of string (hex data)
  - storageProofs: array of object
    - key: string (hex data)
    - proof: array of string (hex data)
    - value: object
      - hasValue: boolean
      - value: object
        - isEmpty: boolean
        - length: string (hex integer)
        - span: object
          - isEmpty: boolean
          - item: object
          - length: string (hex integer)
  - storageRoot: string (hash)
```

</TabItem>
</Tabs>

ethgetStorageAt

Returns storage data at address. storageindex

<Tabs>
<TabItem value="params" label="Parameters">

1. address: string (address)

2. positionIndex: string (hex integer)

3. blockParameter: string (block number or hash or either of earliest, finalized, latest, pending, or safe)

</TabItem>
<TabItem value="request" label="Request" default>

bash
```
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "ethgetStorageAt",
      "params": [address, positionIndex, blockParameter]
    }'
```


</TabItem>
<TabItem value="response" label="Response">

json
```
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}
```


result: string (hex data)

</TabItem>
</Tabs>

ethgetTransactionByBlockHashAndIndex

Retrieves a transaction by block hash and index

<Tabs>
<TabItem value="params" label="Parameters">

1. blockHash: string (hash)

2. positionIndex: string (hex integer)

</TabItem>
<TabItem value="request" label="Request" default>

```
bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "ethgetTransactionByBlockHashAndIndex",
      "params": [blockHash, positionIndex]
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: object
  - accessList: array of object
    - address: string (address)
    - storageKeys: array of string (hex integer)
  - blobVersionedHashes: array of string (hex data)
  - blockHash: string (hash)
  - blockNumber: string (hex integer)
  - chainId: string (hex integer)
  - data: string (hex data)
  - from: string (address)
  - gas: string (hex integer)
  - gasPrice: string (hex integer)
  - hash: string (hash)
  - input: string (hex data)
  - isSystemTx: boolean
  - maxFeePerBlobGas: string (hex integer)
  - maxFeePerGas: string (hex integer)
  - maxPriorityFeePerGas: string (hex integer)
  - mint: string (hex integer)
  - nonce: string (hex integer)
  - r: string (hex integer)
  - s: string (hex integer)
  - sourceHash: string (hash)
  - to: string (address)
  - transactionIndex: string (hex integer)
  - type: integer
  - v: string (hex integer)
  - value: string (hex integer)
  - yParity: string (hex integer)

</TabItem>
</Tabs>

ethgetTransactionByBlockNumberAndIndex

Retrieves a transaction by block number and index

<Tabs>
<TabItem value="params" label="Parameters">

1. blockParameter: string (block number or hash or either of earliest,
```

```
finalized, latest, pending, or safe)

2. positionIndex: string (hex integer)


</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "ethgetTransactionByBlockNumberAndIndex",
      "params": [blockParameter, positionIndex]
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: object
  - accessList: array of object
    - address: string (address)
    - storageKeys: array of string (hex integer)
  - blobVersionedHashes: array of string (hex data)
  - blockHash: string (hash)
  - blockNumber: string (hex integer)
  - chainId: string (hex integer)
  - data: string (hex data)
  - from: string (address)
  - gas: string (hex integer)
  - gasPrice: string (hex integer)
  - hash: string (hash)
  - input: string (hex data)
  - isSystemTx: boolean
  - maxFeePerBlobGas: string (hex integer)
  - maxFeePerGas: string (hex integer)
  - maxPriorityFeePerGas: string (hex integer)
  - mint: string (hex integer)
  - nonce: string (hex integer)
  - r: string (hex integer)
  - s: string (hex integer)
  - sourceHash: string (hash)
  - to: string (address)
  - transactionIndex: string (hex integer)
  - type: integer
  - v: string (hex integer)
  - value: string (hex integer)
  - yParity: string (hex integer)

</TabItem>
</Tabs>
```

ethgetTransactionByHash

Retrieves a transaction by hash

<Tabs>
<TabItem value="params" label="Parameters">

1. transactionHash: string (hash)

</TabItem>
<TabItem value="request" label="Request" default>

bash
```
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "ethgetTransactionByHash",
      "params": [transactionHash]
    }'
```


</TabItem>
<TabItem value="response" label="Response">

json
```
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}
```


result: object
- accessList: array of object
  - address: string (address)
  - storageKeys: array of string (hex integer)
- blobVersionedHashes: array of string (hex data)
- blockHash: string (hash)
- blockNumber: string (hex integer)
- chainId: string (hex integer)
- data: string (hex data)
- from: string (address)
- gas: string (hex integer)
- gasPrice: string (hex integer)
- hash: string (hash)
- input: string (hex data)
- isSystemTx: boolean
- maxFeePerBlobGas: string (hex integer)
- maxFeePerGas: string (hex integer)
- maxPriorityFeePerGas: string (hex integer)
- mint: string (hex integer)
- nonce: string (hex integer)
- r: string (hex integer)
- s: string (hex integer)
- sourceHash: string (hash)
- to: string (address)
- transactionIndex: string (hex integer)
- type: integer
- v: string (hex integer)
- value: string (hex integer)

```
  - yParity: string (hex integer)

</TabItem>
</Tabs>

ethgetTransactionCount

Returns account nonce (number of trnsactions from the account since genesis) at
the given block number

<Tabs>
<TabItem value="params" label="Parameters">

1. address: string (address)

2. blockParameter: string (block number or hash or either of earliest,
finalized, latest, pending, or safe)


</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "ethgetTransactionCount",
      "params": [address, blockParameter]
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: string (hex integer)

</TabItem>
</Tabs>

ethgetTransactionReceipt

Retrieves a transaction receipt by tx hash

<Tabs>
<TabItem value="params" label="Parameters">

1. txHashData: string (hash)


</TabItem>
<TabItem value="request" label="Request" default>

bash
```

```
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "ethgetTransactionReceipt",
      "params": [txHashData]
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: object
  - blobGasPrice: string (hex integer)
  - blobGasUsed: string (hex integer)
  - blockHash: string (hash)
  - blockNumber: string (hex integer)
  - contractAddress: string (address)
  - cumulativeGasUsed: string (hex integer)
  - effectiveGasPrice: string (hex integer)
  - error: string
  - from: string (address)
  - gasUsed: string (hex integer)
  - logs: array of object
    - address: string (address)
    - blockHash: string (hash)
    - blockNumber: string (hex integer)
    - data: string (hex data)
    - logIndex: string (hex integer)
    - removed: boolean
    - topics: array of string (hash)
    - transactionHash: string (hash)
    - transactionIndex: string (hex integer)
  - logsBloom: string (hex data)
  - root: string (hash)
  - status: string (hex integer)
  - to: string (address)
  - transactionHash: string (hash)
  - transactionIndex: string (hex integer)
  - type: integer

</TabItem>
</Tabs>

ethgetUncleByBlockHashAndIndex

Retrieves an uncle block header by block hash and uncle index

<Tabs>
<TabItem value="params" label="Parameters">

1. blockHashData: string (hash)

2. positionIndex: string (hex integer)
```

```
</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "ethgetUncleByBlockHashAndIndex",
      "params": [blockHashData, positionIndex]
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: object
  - author: string (address)
  - baseFeePerGas: string (hex integer)
  - blobGasUsed: string (hex integer)
  - difficulty: string (hex integer)
  - excessBlobGas: string (hex integer)
  - extraData: string (hex data)
  - gasLimit: string (hex integer)
  - gasUsed: string (hex integer)
  - hash: string (hash)
  - logsBloom: string (hex data)
  - miner: string (address)
  - mixHash: string (hash)
  - nonce: string (hex data)
  - number: string (hex integer)
  - parentBeaconBlockRoot: string (hash)
  - parentHash: string (hash)
  - receiptsRoot: string (hash)
  - sha3Uncles: string (hash)
  - signature: string (hex data)
  - size: string (hex integer)
  - stateRoot: string (hash)
  - step: string (hex integer)
  - timestamp: string (hex integer)
  - totalDifficulty: string (hex integer)
  - transactions: array of object
  - transactionsRoot: string (hash)
  - uncles: array of string (hash)
  - withdrawals: array of object
    - address: string (address)
    - amountInGwei: string (hex integer)
    - amountInWei: string (hex integer)
    - index: string (hex integer)
    - validatorIndex: string (hex integer)
  - withdrawalsRoot: string (hash)
```

</TabItem>
</Tabs>

ethgetUncleByBlockNumberAndIndex

Retrieves an uncle block header by block number and uncle index

<Tabs>
<TabItem value="params" label="Parameters">

1. blockParameter: string (block number or hash or either of earliest, finalized, latest, pending, or safe)

2. positionIndex: string (hex integer)

</TabItem>
<TabItem value="request" label="Request" default>

bash
```
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "ethgetUncleByBlockNumberAndIndex",
      "params": [blockParameter, positionIndex]
    }'
```

</TabItem>
<TabItem value="response" label="Response">

json
```
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}
```

result: object
- author: string (address)
- baseFeePerGas: string (hex integer)
- blobGasUsed: string (hex integer)
- difficulty: string (hex integer)
- excessBlobGas: string (hex integer)
- extraData: string (hex data)
- gasLimit: string (hex integer)
- gasUsed: string (hex integer)
- hash: string (hash)
- logsBloom: string (hex data)
- miner: string (address)
- mixHash: string (hash)
- nonce: string (hex data)
- number: string (hex integer)
- parentBeaconBlockRoot: string (hash)
- parentHash: string (hash)
- receiptsRoot: string (hash)
- sha3Uncles: string (hash)
- signature: string (hex data)
- size: string (hex integer)
- stateRoot: string (hash)

```
  - step: string (hex integer)
  - timestamp: string (hex integer)
  - totalDifficulty: string (hex integer)
  - transactions: array of object
  - transactionsRoot: string (hash)
  - uncles: array of string (hash)
  - withdrawals: array of object
    - address: string (address)
    - amountInGwei: string (hex integer)
    - amountInWei: string (hex integer)
    - index: string (hex integer)
    - validatorIndex: string (hex integer)
  - withdrawalsRoot: string (hash)

</TabItem>
</Tabs>

ethgetUncleCountByBlockHash

Returns number of uncles in the block by block hash

<Tabs>
<TabItem value="params" label="Parameters">

1. blockHash: string (hash)


</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "ethgetUncleCountByBlockHash",
      "params": [blockHash]
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: string (hex integer)

</TabItem>
</Tabs>

ethgetUncleCountByBlockNumber

Returns number of uncles in the block by block number

<Tabs>
<TabItem value="params" label="Parameters">
```

1. blockParameter: string (block number or hash or either of earliest, finalized, latest, pending, or safe)

</TabItem>
<TabItem value="request" label="Request" default>

```
bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "ethgetUncleCountByBlockNumber",
      "params": [blockParameter]
    }'
```

</TabItem>
<TabItem value="response" label="Response">

```
json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}
```

result: string (hex integer)

</TabItem>
</Tabs>

ethnewBlockFilter

Creates an update filter

<Tabs>
<TabItem value="request" label="Request" default>

```
bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "ethnewBlockFilter",
      "params": []
    }'
```

</TabItem>
<TabItem value="response" label="Response">

```
json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}
```

```
result: string (hex integer)

</TabItem>
</Tabs>

ethnewFilter

Creates an update filter

<Tabs>
<TabItem value="params" label="Parameters">

1. filter: object
    - address: object
    - fromBlock: string (block number or hash or either of earliest, finalized,
latest, pending, or safe)
    - toBlock: string (block number or hash or either of earliest, finalized,
latest, pending, or safe)
    - topics: array of object


</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "ethnewFilter",
      "params": [filter]
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: string (hex integer)

</TabItem>
</Tabs>

ethnewPendingTransactionFilter

Creates an update filter

<Tabs>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
```

```
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "ethnewPendingTransactionFilter",
      "params": []
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: string (hex integer)

</TabItem>
</Tabs>

ethpendingTransactions

Returns the pending transactions list

<Tabs>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "ethpendingTransactions",
      "params": []
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: array of object
  - accessList: array of object
    - address: string (address)
    - storageKeys: array of string (hex integer)
  - blobVersionedHashes: array of string (hex data)
  - blockHash: string (hash)
  - blockNumber: string (hex integer)
  - chainId: string (hex integer)
  - data: string (hex data)
```

```
  - from: string (address)
  - gas: string (hex integer)
  - gasPrice: string (hex integer)
  - hash: string (hash)
  - input: string (hex data)
  - isSystemTx: boolean
  - maxFeePerBlobGas: string (hex integer)
  - maxFeePerGas: string (hex integer)
  - maxPriorityFeePerGas: string (hex integer)
  - mint: string (hex integer)
  - nonce: string (hex integer)
  - r: string (hex integer)
  - s: string (hex integer)
  - sourceHash: string (hash)
  - to: string (address)
  - transactionIndex: string (hex integer)
  - type: integer
  - v: string (hex integer)
  - value: string (hex integer)
  - yParity: string (hex integer)

</TabItem>
</Tabs>

ethprotocolVersion

Returns ETH protocol version

<Tabs>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "ethprotocolVersion",
      "params": []
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: string

</TabItem>
</Tabs>

ethsendRawTransaction

Send a raw transaction to the tx pool and broadcasting

<Tabs>
```

<TabItem value="params" label="Parameters">

1. transaction: string (hex data)

</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "ethsendRawTransaction",
      "params": [transaction]
    }'

</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}

result: string (hash)

</TabItem>
</Tabs>

ethsendTransaction

Send a transaction to the tx pool and broadcasting

<Tabs>
<TabItem value="params" label="Parameters">

1. rpcTx: object
    - accessList: array of object
      - address: string (address)
      - storageKeys: array of string (hex integer)
    - blobVersionedHashes: array of string (hex data)
    - blockHash: string (hash)
    - blockNumber: string (hex integer)
    - chainId: string (hex integer)
    - data: string (hex data)
    - from: string (address)
    - gas: string (hex integer)
    - gasPrice: string (hex integer)
    - hash: string (hash)
    - input: string (hex data)
    - isSystemTx: boolean
    - maxFeePerBlobGas: string (hex integer)
    - maxFeePerGas: string (hex integer)
    - maxPriorityFeePerGas: string (hex integer)
    - mint: string (hex integer)
    - nonce: string (hex integer)
    - r: string (hex integer)

```
    - s: string (hex integer)
    - sourceHash: string (hash)
    - to: string (address)
    - transactionIndex: string (hex integer)
    - type: integer
    - v: string (hex integer)
    - value: string (hex integer)
    - yParity: string (hex integer)


</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "ethsendTransaction",
      "params": [rpcTx]
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: string (hash)

</TabItem>
</Tabs>

ethsimulateV1

Executes a simulation across multiple blocks (does not create a transaction or
block)

<Tabs>
<TabItem value="params" label="Parameters">

1. payload: object
    - blockStateCalls: array of object
      - blockOverrides: object
        - baseFeePerGas: string (hex integer)
        - blobBaseFee: string (hex integer)
        - feeRecipient: string (address)
        - gasLimit: string (hex integer)
        - number: string (hex integer)
        - prevRandao: string (hash)
        - time: string (hex integer)
      - calls: array of object
        - accessList: array of object
          - address: string (address)
          - storageKeys: array of string (hex integer)
        - blobVersionedHashes: array of string (hex data)
```

```
        - blockHash: string (hash)
        - blockNumber: string (hex integer)
        - chainId: string (hex integer)
        - data: string (hex data)
        - from: string (address)
        - gas: string (hex integer)
        - gasPrice: string (hex integer)
        - hash: string (hash)
        - input: string (hex data)
        - isSystemTx: boolean
        - maxFeePerBlobGas: string (hex integer)
        - maxFeePerGas: string (hex integer)
        - maxPriorityFeePerGas: string (hex integer)
        - mint: string (hex integer)
        - nonce: string (hex integer)
        - r: string (hex integer)
        - s: string (hex integer)
        - sourceHash: string (hash)
        - to: string (address)
        - transactionIndex: string (hex integer)
        - type: integer
        - v: string (hex integer)
        - value: string (hex integer)
        - yParity: string (hex integer)
      - stateOverrides: map of object
        - balance: string (hex integer)
        - code: string (hex data)
        - movePrecompileToAddress: string (address)
        - nonce: string (hex integer)
        - state: map of string (hash)
        - stateDiff: map of string (hash)
    - returnFullTransactionObjects: boolean
    - traceTransfers: boolean
    - validation: boolean

2. blockParameter: string (block number or hash or either of earliest,
finalized, latest, pending, or safe)


</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "ethsimulateV1",
      "params": [payload, blockParameter]
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}
```

```
result: array of object
  - calls: array of object
    - error: object
      - code: string (hex integer)
      - data: string
      - message: string
    - gasUsed: string (hex integer)
    - logs: array of object
      - address: string (address)
      - blockHash: string (hash)
      - blockNumber: string (hex integer)
      - data: string (hex data)
      - logIndex: string (hex integer)
      - removed: boolean
      - topics: array of string (hash)
      - transactionHash: string (hash)
      - transactionIndex: string (hex integer)
    - returnData: string (hex data)
    - status: string (hex integer)

</TabItem>
</Tabs>

ethsubscribe

Starts a subscription to a particular event over WebSockets. A JSON-RPC
notification with event payload and subscription id is sent to a client for
every event matching the subscription topic.

:::info
This method is enabled by adding subscribe to --JsonRpc.EnabledModules.
:::

<Tabs>
<TabItem value="params" label="Parameters">

1. subscriptionName: string

2. filter: object
    - address: string (address)
    - fromBlock: string (block number or hash or either of earliest, finalized,
latest, pending, or safe)
    - toBlock: string (block number or hash or either of earliest, finalized,
latest, pending, or safe)
    - topics: array of string (hex data)

</TabItem>
<TabItem value="request" label="Request" default>

bash
wscat -c ws://localhost:8545


json
{
  "jsonrpc": "2.0",
  "id": 0,
  "method": "ethsubscribe",
  "params": [subscriptionName, args]
}


</TabItem>
```

<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": "0x..." // subscription id
}

result: string

</TabItem>
<TabItem value="notif" label="Notification">

json
{
  "jsonrpc": "2.0",
  "method": "ethsubscription",
  "params": {
    "subscription": "0x...", // subscription id
    "result": payload
  }
}

See specific subcription topic below for payload details.

</TabItem>
</Tabs>

Subscription topics

<details>
<summary className="nd-details-heading">

newHeads

</summary>
<p>

Subscribes to incoming block headers. Fires a notification each time a new header is appended to the chain, including chain reorganizations.

Notification payload: object
  - author: string (address)
  - baseFeePerGas: string (hex integer)
  - blobGasUsed: string (hex integer)
  - difficulty: string (hex integer)
  - excessBlobGas: string (hex integer)
  - extraData: string (hex data)
  - gasLimit: string (hex integer)
  - gasUsed: string (hex integer)
  - hash: string (hash)
  - logsBloom: string (hex data)
  - miner: string (address)
  - mixHash: string (hash)
  - nonce: string (hex data)
  - number: string (hex integer)
  - parentBeaconBlockRoot: string (hash)
  - parentHash: string (hash)
  - receiptsRoot: string (hash)
  - sha3Uncles: string (hash)

```
  - signature: string (hex data)
  - size: string (hex integer)
  - stateRoot: string (hash)
  - step: string (hex integer)
  - timestamp: string (hex integer)
  - totalDifficulty: string (hex integer)
  - transactions: array of object
  - transactionsRoot: string (hash)
  - uncles: array of string (hash)
  - withdrawals: array of object
    - address: string (address)
    - amount: string (hex integer)
    - index: string (hex integer)
    - validatorIndex: string (hex integer)
  - withdrawalsRoot: string (hash)

</p>
</details>

<details>
<summary className="nd-details-heading">

logs

</summary>
<p>

Subscribes to incoming logs filtered by the given options. In case of a chain
reorganization, previously sent logs on the old chain will be re-sent with the
removed field set to true.

Notification payload: object
  - address: string (address)
  - blockHash: string (hash)
  - blockNumber: string (hex integer)
  - data: string (hex data)
  - logIndex: string (hex integer)
  - removed: boolean
  - topics: array of string (hash)
  - transactionHash: string (hash)
  - transactionIndex: string (hex integer)
  - transactionLogIndex: string (hex integer)

</p>
</details>

<details>
<summary className="nd-details-heading">

newPendingTransactions

</summary>
<p>

Subscribes to incoming pending transactions. Returns the transaction hash.

Notification payload: string (hash)

</p>
</details>

<details>
<summary className="nd-details-heading">
```

```
droppedPendingTransactions

</summary>
<p>

Subscribes to transactions evicted from the transaction pool. Returns the
transaction hash.

Notification payload: string (hash)

</p>
</details>

<details>
<summary className="nd-details-heading">

syncing

</summary>
<p>

Subscribes to syncing events. Returns false (once) if the node is synced or an
object with statistics (once) when the node starts syncing.

Notification payload:

  - if synced: boolean
  - if syncing: object
      - currentBlock: string (hex integer)
      - highestBlock: string (hex integer)
      - isSyncing: boolean
      - startingBlock: string (hex integer)

</p>
</details>
ethsyncing

Returns syncing status

<Tabs>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "ethsyncing",
      "params": []
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}
```

```
result: object
  - currentBlock: string (hex integer)
  - highestBlock: string (hex integer)
  - isSyncing: boolean
  - startingBlock: string (hex integer)
  - syncMode: integer

</TabItem>
</Tabs>

ethuninstallFilter

Creates an update filter

<Tabs>
<TabItem value="params" label="Parameters">

1. filterId: string (hex integer)


</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "ethuninstallFilter",
      "params": [filterId]
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: boolean

</TabItem>
</Tabs>

ethunsubscribe

Unsubscribes from a subscription.

:::info
This method is enabled by adding subscribe to --JsonRpc.EnabledModules.
:::

<Tabs>
<TabItem value="params" label="Parameters">

1. subscriptionId: string
```

</TabItem>
<TabItem value="request" label="Request" default>

bash
wscat -c ws://localhost:8545

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "method": "ethunsubscribe",
  "params": [subscriptionId]
}

</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}

result: boolean (true if unsubscribed successfully; otherwise, false)

</TabItem>
</Tabs>

eth_subscribe.md:

ethsubscribe

Starts a subscription to a particular event over WebSockets. A JSON-RPC notification with event payload and subscription id is sent to a client for every event matching the subscription topic.

:::info
This method is enabled by adding subscribe to --JsonRpc.EnabledModules.
:::

<Tabs>
<TabItem value="params" label="Parameters">

1. subscriptionName: string

2. filter: object
    - address: string (address)
    - fromBlock: string (block number or hash or either of earliest, finalized, latest, pending, or safe)
    - toBlock: string (block number or hash or either of earliest, finalized, latest, pending, or safe)
    - topics: array of string (hex data)

</TabItem>
<TabItem value="request" label="Request" default>

bash

```
wscat -c ws://localhost:8545


json
{
  "jsonrpc": "2.0",
  "id": 0,
  "method": "ethsubscribe",
  "params": [subscriptionName, args]
}


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": "0x..." // subscription id
}


result: string

</TabItem>
<TabItem value="notif" label="Notification">


json
{
  "jsonrpc": "2.0",
  "method": "ethsubscription",
  "params": {
    "subscription": "0x...", // subscription id
    "result": payload
  }
}


See specific subcription topic below for payload details.

</TabItem>
</Tabs>

Subscription topics

<details>
<summary className="nd-details-heading">

newHeads

</summary>
<p>

Subscribes to incoming block headers. Fires a notification each time a new
header is appended to the chain, including chain reorganizations.

Notification payload: object
  - author: string (address)
  - baseFeePerGas: string (hex integer)
  - blobGasUsed: string (hex integer)
  - difficulty: string (hex integer)
  - excessBlobGas: string (hex integer)
```

  - extraData: string (hex data)
  - gasLimit: string (hex integer)
  - gasUsed: string (hex integer)
  - hash: string (hash)
  - logsBloom: string (hex data)
  - miner: string (address)
  - mixHash: string (hash)
  - nonce: string (hex data)
  - number: string (hex integer)
  - parentBeaconBlockRoot: string (hash)
  - parentHash: string (hash)
  - receiptsRoot: string (hash)
  - sha3Uncles: string (hash)
  - signature: string (hex data)
  - size: string (hex integer)
  - stateRoot: string (hash)
  - step: string (hex integer)
  - timestamp: string (hex integer)
  - totalDifficulty: string (hex integer)
  - transactions: array of object
  - transactionsRoot: string (hash)
  - uncles: array of string (hash)
  - withdrawals: array of object
    - address: string (address)
    - amount: string (hex integer)
    - index: string (hex integer)
    - validatorIndex: string (hex integer)
  - withdrawalsRoot: string (hash)

</details>

<details>
<summary className="nd-details-heading">

logs

</summary>

Subscribes to incoming logs filtered by the given options. In case of a chain reorganization, previously sent logs on the old chain will be re-sent with the removed field set to true.

Notification payload: object
  - address: string (address)
  - blockHash: string (hash)
  - blockNumber: string (hex integer)
  - data: string (hex data)
  - logIndex: string (hex integer)
  - removed: boolean
  - topics: array of string (hash)
  - transactionHash: string (hash)
  - transactionIndex: string (hex integer)
  - transactionLogIndex: string (hex integer)

</details>

<details>
<summary className="nd-details-heading">

newPendingTransactions

</summary>

Subscribes to incoming pending transactions. Returns the transaction hash.

Notification payload: string (hash)

</details>

<details>
<summary className="nd-details-heading">

droppedPendingTransactions

</summary>

Subscribes to transactions evicted from the transaction pool. Returns the transaction hash.

Notification payload: string (hash)

</details>

<details>
<summary className="nd-details-heading">

syncing

</summary>

Subscribes to syncing events. Returns false (once) if the node is synced or an object with statistics (once) when the node starts syncing.

Notification payload:

  - if synced: boolean
  - if syncing: object
      - currentBlock: string (hex integer)
      - highestBlock: string (hex integer)
      - isSyncing: boolean
      - startingBlock: string (hex integer)

</details>

eth_unsubscribe.md:

ethunsubscribe

Unsubscribes from a subscription.

:::info
This method is enabled by adding subscribe to --JsonRpc.EnabledModules.
:::

<Tabs>
<TabItem value="params" label="Parameters">

1. subscriptionId: string

```
</TabItem>
<TabItem value="request" label="Request" default>

bash
wscat -c ws://localhost:8545


json
{
  "jsonrpc": "2.0",
  "id": 0,
  "method": "ethunsubscribe",
  "params": [subscriptionId]
}


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: boolean (true if unsubscribed successfully; otherwise, false)

</TabItem>
</Tabs>


# net.md:

---
title: net namespace
sidebarlabel: net
sidebarposition: 4
---

import Tabs from "@theme/Tabs";
import TabItem from "@theme/TabItem";

netlistening

<Tabs>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "netlistening",
      "params": []
    }'


</TabItem>
```

<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}

result: boolean

</TabItem>
</Tabs>

netlocalAddress

<Tabs>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "netlocalAddress",
      "params": []
    }'

</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}

result: string (address)

</TabItem>
</Tabs>

netlocalEnode

<Tabs>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "netlocalEnode",
      "params": []
    }'

```
</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: string

</TabItem>
</Tabs>

netpeerCount

<Tabs>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "netpeerCount",
      "params": []
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: string (hex integer)

</TabItem>
</Tabs>

netversion

<Tabs>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "netversion",
      "params": []
```

```
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: string

</TabItem>
</Tabs>



# parity.md:

---
title: parity namespace
sidebarlabel: parity
sidebarposition: 5
---

import Tabs from "@theme/Tabs";
import TabItem from "@theme/TabItem";

parityclearEngineSigner

<Tabs>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "parityclearEngineSigner",
      "params": []
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: boolean

</TabItem>
</Tabs>
```

parityenode

Returns the node enode URI.

<Tabs>
<TabItem value="request" label="Request" default>

bash
```
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "parityenode",
      "params": []
    }'
```

</TabItem>
<TabItem value="response" label="Response">

json
```
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}
```

result: string

</TabItem>
</Tabs>

paritygetBlockReceipts

Get receipts from all transactions from particular block, more efficient than fetching the receipts one-by-one.

<Tabs>
<TabItem value="params" label="Parameters">

1. blockParameter: string (block number or hash or either of earliest, finalized, latest, pending, or safe)

</TabItem>
<TabItem value="request" label="Request" default>

bash
```
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "paritygetBlockReceipts",
      "params": [blockParameter]
    }'
```

</TabItem>

<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}

result: array of object
  - blobGasPrice: string (hex integer)
  - blobGasUsed: string (hex integer)
  - blockHash: string (hash)
  - blockNumber: string (hex integer)
  - contractAddress: string (address)
  - cumulativeGasUsed: string (hex integer)
  - effectiveGasPrice: string (hex integer)
  - error: string
  - from: string (address)
  - gasUsed: string (hex integer)
  - logs: array of object
    - address: string (address)
    - blockHash: string (hash)
    - blockNumber: string (hex integer)
    - data: string (hex data)
    - logIndex: string (hex integer)
    - removed: boolean
    - topics: array of string (hash)
    - transactionHash: string (hash)
    - transactionIndex: string (hex integer)
  - logsBloom: string (hex data)
  - root: string (hash)
  - status: string (hex integer)
  - to: string (address)
  - transactionHash: string (hash)
  - transactionIndex: string (hex integer)
  - type: integer

</TabItem>
</Tabs>

paritynetPeers

Returns connected peers. Peers with non-empty protocols have completed
handshake.

<Tabs>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "paritynetPeers",
      "params": []
    }'

</TabItem>
<TabItem value="response" label="Response">

```
json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: object
  - active: string (hex integer)
  - connected: string (hex integer)
  - max: string (hex integer)
  - peers: array of object
    - caps: array of string
    - id: string
    - name: string
    - network: object
      - localAddress: string
      - remoteAddress: string
    - protocols: map of object
      - difficulty: string (hex integer)
      - headHash: string (hash)
      - version: string (hex data)

</TabItem>
</Tabs>

paritypendingTransactions

Returns a list of transactions currently in the queue. If address is provided,
returns transactions only with given sender address.

<Tabs>
<TabItem value="params" label="Parameters">

1. address: string (address)


</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "paritypendingTransactions",
      "params": [address]
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}
```

```
result: array of object
  - blockHash: string (hash)
  - blockNumber: string (hex integer)
  - chainId: string (hex integer)
  - condition: object
  - creates: string (address)
  - from: string (address)
  - gas: string (hex integer)
  - gasPrice: string (hex integer)
  - hash: string (hash)
  - input: string (hex data)
  - nonce: string (hex integer)
  - publicKey: object
    - address: string (address)
    - bytes: string (hex data)
    - prefixedBytes: string (hex data)
  - r: string (hex data)
  - raw: string (hex data)
  - s: string (hex data)
  - standardV: string (hex integer)
  - to: string (address)
  - transactionIndex: string (hex integer)
  - v: string (hex integer)
  - value: string (hex integer)

</TabItem>
</Tabs>

paritysetEngineSigner

<Tabs>
<TabItem value="params" label="Parameters">

1. address: string (address)

2. password: string


</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "paritysetEngineSigner",
      "params": [address, password]
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}
```

```
result: boolean

</TabItem>
</Tabs>

paritysetEngineSignerSecret

<Tabs>
<TabItem value="params" label="Parameters">

1. privateKey: string


</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "paritysetEngineSignerSecret",
      "params": [privateKey]
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: boolean

</TabItem>
</Tabs>



# personal.md:

---
title: personal namespace
sidebarlabel: personal
sidebarposition: 6
---

import Tabs from "@theme/Tabs";
import TabItem from "@theme/TabItem";

personalimportRawKey

<Tabs>
<TabItem value="params" label="Parameters">

1. keyData: string (hex data)
```

2. passphrase: string

</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "personalimportRawKey",
      "params": [keyData, passphrase]
    }'

</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}

result: string (address)

</TabItem>
</Tabs>

personallistAccounts

<Tabs>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "personallistAccounts",
      "params": []
    }'

</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}

result: array of string (address)

```
</TabItem>
</Tabs>

personallockAccount

<Tabs>
<TabItem value="params" label="Parameters">

1. address: string (address)


</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "personallockAccount",
      "params": [address]
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: boolean

</TabItem>
</Tabs>

personalnewAccount

<Tabs>
<TabItem value="params" label="Parameters">

1. passphrase: string


</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "personalnewAccount",
      "params": [passphrase]
    }'
```

```
</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: string (address)

</TabItem>
</Tabs>

personalunlockAccount

<Tabs>
<TabItem value="params" label="Parameters">

1. address: string (address)

2. passphrase: string


</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "personalunlockAccount",
      "params": [address, passphrase]
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: boolean

</TabItem>
</Tabs>



# proof.md:

---
```

```
title: proof namespace
sidebarlabel: proof
sidebarposition: 7
---

import Tabs from "@theme/Tabs";
import TabItem from "@theme/TabItem";

proofgetTransactionByHash

This function returns the same result as ethgetTransactionReceipt and also a tx
proof, receipt proof and serialized block headers.

<Tabs>
<TabItem value="params" label="Parameters">

1. txHash: string (hash)

2. includeHeader: boolean


</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "proofgetTransactionByHash",
      "params": [txHash, includeHeader]
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: object
  - blockHeader: string (hex data)
  - transaction: object
    - accessList: array of object
      - address: string (address)
      - storageKeys: array of string (hex integer)
    - blobVersionedHashes: array of string (hex data)
    - blockHash: string (hash)
    - blockNumber: string (hex integer)
    - chainId: string (hex integer)
    - data: string (hex data)
    - from: string (address)
    - gas: string (hex integer)
    - gasPrice: string (hex integer)
    - hash: string (hash)
    - input: string (hex data)
    - isSystemTx: boolean
```

    - maxFeePerBlobGas: string (hex integer)
    - maxFeePerGas: string (hex integer)
    - maxPriorityFeePerGas: string (hex integer)
    - mint: string (hex integer)
    - nonce: string (hex integer)
    - r: string (hex integer)
    - s: string (hex integer)
    - sourceHash: string (hash)
    - to: string (address)
    - transactionIndex: string (hex integer)
    - type: integer
    - v: string (hex integer)
    - value: string (hex integer)
    - yParity: string (hex integer)
  - txProof: array of string (hex data)

</TabItem>
</Tabs>

proofgetTransactionReceipt

This function should return the same result as ethcall and also proofs of all used accounts and their storages and serialized block headers.

<Tabs>
<TabItem value="params" label="Parameters">

1. txHash: string (hash)

2. includeHeader: boolean

</TabItem>
<TabItem value="request" label="Request" default>

bash
```
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "proofgetTransactionReceipt",
      "params": [txHash, includeHeader]
    }'
```


</TabItem>
<TabItem value="response" label="Response">

json
```
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}
```


result: object
  - blockHeader: string (hex data)
  - receipt: object
    - blobGasPrice: string (hex integer)
    - blobGasUsed: string (hex integer)
    - blockHash: string (hash)

```
    - blockNumber: string (hex integer)
    - contractAddress: string (address)
    - cumulativeGasUsed: string (hex integer)
    - effectiveGasPrice: string (hex integer)
    - error: string
    - from: string (address)
    - gasUsed: string (hex integer)
    - logs: array of object
      - address: string (address)
      - blockHash: string (hash)
      - blockNumber: string (hex integer)
      - data: string (hex data)
      - logIndex: string (hex integer)
      - removed: boolean
      - topics: array of string (hash)
      - transactionHash: string (hash)
      - transactionIndex: string (hex integer)
    - logsBloom: string (hex data)
    - root: string (hash)
    - status: string (hex integer)
    - to: string (address)
    - transactionHash: string (hash)
    - transactionIndex: string (hex integer)
    - type: integer
  - receiptProof: array of string (hex data)
  - txProof: array of string (hex data)

</TabItem>
</Tabs>



# trace.md:

---
title: trace namespace
sidebarlabel: trace
sidebarposition: 8
---

import Tabs from "@theme/Tabs";
import TabItem from "@theme/TabItem";

traceblock

<Tabs>
<TabItem value="params" label="Parameters">

1. numberOrTag: string (block number or hash or either of earliest, finalized,
latest, pending, or safe)


</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "traceblock",
      "params": [numberOrTag]
```

```
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: array of object
  - action: object
    - author: string (address)
    - callType: string
    - creationMethod: string
    - error: string
    - from: string (address)
    - gas: string (hex integer)
    - includeInTrace: boolean
    - input: string (hex data)
    - isPrecompiled: boolean
    - result: object
      - address: string (address)
      - code: string (hex data)
      - gasUsed: string (hex integer)
      - output: string (hex data)
    - rewardType: string
    - subtraces: array of object
      <!--[circular ref]-->
    - to: string (address)
    - traceAddress: array of string (hex integer)
    - type: string
    - value: string (hex integer)
  - blockHash: string (hash)
  - blockNumber: string (hex integer)
  - error: string
  - result: object
    - address: string (address)
    - code: string (hex data)
    - gasUsed: string (hex integer)
    - output: string (hex data)
  - subtraces: string (hex integer)
  - traceAddress: array of string (hex integer)
  - transactionHash: string (hash)
  - transactionPosition: string (hex integer)
  - type: string

</TabItem>
</Tabs>

tracecall

<Tabs>
<TabItem value="params" label="Parameters">

1. call: object
    - accessList: array of object
      - address: string (address)
      - storageKeys: array of string (hex integer)
    - blobVersionedHashes: array of string (hex data)
```

```
    - blockHash: string (hash)
    - blockNumber: string (hex integer)
    - chainId: string (hex integer)
    - data: string (hex data)
    - from: string (address)
    - gas: string (hex integer)
    - gasPrice: string (hex integer)
    - hash: string (hash)
    - input: string (hex data)
    - isSystemTx: boolean
    - maxFeePerBlobGas: string (hex integer)
    - maxFeePerGas: string (hex integer)
    - maxPriorityFeePerGas: string (hex integer)
    - mint: string (hex integer)
    - nonce: string (hex integer)
    - r: string (hex integer)
    - s: string (hex integer)
    - sourceHash: string (hash)
    - to: string (address)
    - transactionIndex: string (hex integer)
    - type: integer
    - v: string (hex integer)
    - value: string (hex integer)
    - yParity: string (hex integer)

2. traceTypes: array of string

3. blockParameter: string (block number or hash or either of earliest,
finalized, latest, pending, or safe)


</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "tracecall",
      "params": [call, traceTypes, blockParameter]
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: object
  - action: object
    - author: string (address)
    - callType: string
    - creationMethod: string
    - error: string
    - from: string (address)
```

```
    - gas: string (hex integer)
    - includeInTrace: boolean
    - input: string (hex data)
    - isPrecompiled: boolean
    - result: object
      - address: string (address)
      - code: string (hex data)
      - gasUsed: string (hex integer)
      - output: string (hex data)
    - rewardType: string
    - subtraces: array of object
      <!--[circular ref]-->
    - to: string (address)
    - traceAddress: array of string (hex integer)
    - type: string
    - value: string (hex integer)
  - output: string (hex data)
  - stateChanges: map of object
    - balance: object
      - after: string (hex integer)
      - before: string (hex integer)
    - code: object
      - after: string (hex data)
      - before: string (hex data)
    - nonce: object
      - after: string (hex integer)
      - before: string (hex integer)
    - storage: map of object
      - after: string (hex data)
      - before: string (hex data)
  - transactionHash: string (hash)
  - vmTrace: object
    - code: string (hex data)
    - operations: array of object
      - cost: string (hex integer)
      - memory: object
        - data: string (hex data)
        - offset: string (hex integer)
      - pc: string (hex integer)
      - push: array of string (hex data)
      - store: object
        - key: string (hex data)
        - value: string (hex data)
      - sub: object
        <!--[circular ref]-->
      - used: string (hex integer)

</TabItem>
</Tabs>

tracefilter

<Tabs>
<TabItem value="params" label="Parameters">

1. traceFilterForRpc: object
    - after: string (hex integer)
    - count: string (hex integer)
    - fromAddress: array of string (address)
    - fromBlock: string (block number or hash or either of earliest, finalized,
latest, pending, or safe)
    - toAddress: array of string (address)
    - toBlock: string (block number or hash or either of earliest, finalized,
latest, pending, or safe)
```

```
</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "tracefilter",
      "params": [traceFilterForRpc]
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: array of object
  - action: object
    - author: string (address)
    - callType: string
    - creationMethod: string
    - error: string
    - from: string (address)
    - gas: string (hex integer)
    - includeInTrace: boolean
    - input: string (hex data)
    - isPrecompiled: boolean
    - result: object
      - address: string (address)
      - code: string (hex data)
      - gasUsed: string (hex integer)
      - output: string (hex data)
    - rewardType: string
    - subtraces: array of object
      <!--[circular ref]-->
    - to: string (address)
    - traceAddress: array of string (hex integer)
    - type: string
    - value: string (hex integer)
  - blockHash: string (hash)
  - blockNumber: string (hex integer)
  - error: string
  - result: object
    - address: string (address)
    - code: string (hex data)
    - gasUsed: string (hex integer)
    - output: string (hex data)
  - subtraces: string (hex integer)
  - traceAddress: array of string (hex integer)
  - transactionHash: string (hash)
  - transactionPosition: string (hex integer)
  - type: string
```

</TabItem>
</Tabs>

tracerawTransaction

Traces a call to ethsendRawTransaction without making the call, returning the traces

<Tabs>
<TabItem value="params" label="Parameters">

1. data: string (hex data)

2. traceTypes: array of string

</TabItem>
<TabItem value="request" label="Request" default>

bash
```
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "tracerawTransaction",
      "params": [data, traceTypes]
    }'
```


</TabItem>
<TabItem value="response" label="Response">

json
```
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}
```


result: object
- action: object
  - author: string (address)
  - callType: string
  - creationMethod: string
  - error: string
  - from: string (address)
  - gas: string (hex integer)
  - includeInTrace: boolean
  - input: string (hex data)
  - isPrecompiled: boolean
  - result: object
    - address: string (address)
    - code: string (hex data)
    - gasUsed: string (hex integer)
    - output: string (hex data)
  - rewardType: string
  - subtraces: array of object
    <!--[circular ref]-->
  - to: string (address)
  - traceAddress: array of string (hex integer)

```
    - type: string
    - value: string (hex integer)
  - output: string (hex data)
  - stateChanges: map of object
    - balance: object
      - after: string (hex integer)
      - before: string (hex integer)
    - code: object
      - after: string (hex data)
      - before: string (hex data)
    - nonce: object
      - after: string (hex integer)
      - before: string (hex integer)
    - storage: map of object
      - after: string (hex data)
      - before: string (hex data)
  - transactionHash: string (hash)
  - vmTrace: object
    - code: string (hex data)
    - operations: array of object
      - cost: string (hex integer)
      - memory: object
        - data: string (hex data)
        - offset: string (hex integer)
      - pc: string (hex integer)
      - push: array of string (hex data)
      - store: object
        - key: string (hex data)
        - value: string (hex data)
      - sub: object
        <!--[circular ref]-->
      - used: string (hex integer)

</TabItem>
</Tabs>

traceReplayBlockTransactions

<Tabs>
<TabItem value="params" label="Parameters">

1. blockParameter: string (block number or hash or either of earliest,
finalized, latest, pending, or safe)

2. traceTypes: array of string


</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "traceReplayBlockTransactions",
      "params": [blockParameter, traceTypes]
    }'


</TabItem>
<TabItem value="response" label="Response">
```

```
json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: array of object
  - action: object
    - author: string (address)
    - callType: string
    - creationMethod: string
    - error: string
    - from: string (address)
    - gas: string (hex integer)
    - includeInTrace: boolean
    - input: string (hex data)
    - isPrecompiled: boolean
    - result: object
      - address: string (address)
      - code: string (hex data)
      - gasUsed: string (hex integer)
      - output: string (hex data)
    - rewardType: string
    - subtraces: array of object
      <!--[circular ref]-->
    - to: string (address)
    - traceAddress: array of string (hex integer)
    - type: string
    - value: string (hex integer)
  - output: string (hex data)
  - stateChanges: map of object
    - balance: object
      - after: string (hex integer)
      - before: string (hex integer)
    - code: object
      - after: string (hex data)
      - before: string (hex data)
    - nonce: object
      - after: string (hex integer)
      - before: string (hex integer)
    - storage: map of object
      - after: string (hex data)
      - before: string (hex data)
  - transactionHash: string (hash)
  - vmTrace: object
    - code: string (hex data)
    - operations: array of object
      - cost: string (hex integer)
      - memory: object
        - data: string (hex data)
        - offset: string (hex integer)
      - pc: string (hex integer)
      - push: array of string (hex data)
      - store: object
        - key: string (hex data)
        - value: string (hex data)
      - sub: object
        <!--[circular ref]-->
      - used: string (hex integer)

</TabItem>
```

```
</Tabs>

tracereplayTransaction

<Tabs>
<TabItem value="params" label="Parameters">

1. txHash: string (hash)

2. traceTypes: array of string


</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "tracereplayTransaction",
      "params": [txHash, traceTypes]
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: object
  - action: object
    - author: string (address)
    - callType: string
    - creationMethod: string
    - error: string
    - from: string (address)
    - gas: string (hex integer)
    - includeInTrace: boolean
    - input: string (hex data)
    - isPrecompiled: boolean
    - result: object
      - address: string (address)
      - code: string (hex data)
      - gasUsed: string (hex integer)
      - output: string (hex data)
    - rewardType: string
    - subtraces: array of object
      <!--[circular ref]-->
    - to: string (address)
    - traceAddress: array of string (hex integer)
    - type: string
    - value: string (hex integer)
  - output: string (hex data)
  - stateChanges: map of object
    - balance: object
```

```
      - after: string (hex integer)
      - before: string (hex integer)
    - code: object
      - after: string (hex data)
      - before: string (hex data)
    - nonce: object
      - after: string (hex integer)
      - before: string (hex integer)
    - storage: map of object
      - after: string (hex data)
      - before: string (hex data)
  - transactionHash: string (hash)
  - vmTrace: object
    - code: string (hex data)
    - operations: array of object
      - cost: string (hex integer)
      - memory: object
        - data: string (hex data)
        - offset: string (hex integer)
      - pc: string (hex integer)
      - push: array of string (hex data)
      - store: object
        - key: string (hex data)
        - value: string (hex data)
      - sub: object
        <!--[circular ref]-->
      - used: string (hex integer)

</TabItem>
</Tabs>

tracetransaction

<Tabs>
<TabItem value="params" label="Parameters">

1. txHash: string (hash)


</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "tracetransaction",
      "params": [txHash]
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}
```

```
result: array of object
  - action: object
    - author: string (address)
    - callType: string
    - creationMethod: string
    - error: string
    - from: string (address)
    - gas: string (hex integer)
    - includeInTrace: boolean
    - input: string (hex data)
    - isPrecompiled: boolean
    - result: object
      - address: string (address)
      - code: string (hex data)
      - gasUsed: string (hex integer)
      - output: string (hex data)
    - rewardType: string
    - subtraces: array of object
      <!--[circular ref]-->
    - to: string (address)
    - traceAddress: array of string (hex integer)
    - type: string
    - value: string (hex integer)
  - blockHash: string (hash)
  - blockNumber: string (hex integer)
  - error: string
  - result: object
    - address: string (address)
    - code: string (hex data)
    - gasUsed: string (hex integer)
    - output: string (hex data)
  - subtraces: string (hex integer)
  - traceAddress: array of string (hex integer)
  - transactionHash: string (hash)
  - transactionPosition: string (hex integer)
  - type: string

</TabItem>
</Tabs>



# txpool.md:

---
title: txpool namespace
sidebarlabel: txpool
sidebarposition: 9
---

import Tabs from "@theme/Tabs";
import TabItem from "@theme/TabItem";

txpoolcontent

Returns tx pool content.

<Tabs>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
```

```
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "txpoolcontent",
      "params": []
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: object
  - pending: map of map of object
map of object
  - accessList: array of object
    - address: string (address)
    - storageKeys: array of string (hex integer)
  - blobVersionedHashes: array of string (hex data)
  - blockHash: string (hash)
  - blockNumber: string (hex integer)
  - chainId: string (hex integer)
  - data: string (hex data)
  - from: string (address)
  - gas: string (hex integer)
  - gasPrice: string (hex integer)
  - hash: string (hash)
  - input: string (hex data)
  - isSystemTx: boolean
  - maxFeePerBlobGas: string (hex integer)
  - maxFeePerGas: string (hex integer)
  - maxPriorityFeePerGas: string (hex integer)
  - mint: string (hex integer)
  - nonce: string (hex integer)
  - r: string (hex integer)
  - s: string (hex integer)
  - sourceHash: string (hash)
  - to: string (address)
  - transactionIndex: string (hex integer)
  - type: integer
  - v: string (hex integer)
  - value: string (hex integer)
  - yParity: string (hex integer)
  - queued: map of map of object
map of object
  - accessList: array of object
    - address: string (address)
    - storageKeys: array of string (hex integer)
  - blobVersionedHashes: array of string (hex data)
  - blockHash: string (hash)
  - blockNumber: string (hex integer)
  - chainId: string (hex integer)
  - data: string (hex data)
  - from: string (address)
  - gas: string (hex integer)
  - gasPrice: string (hex integer)
```

```
  - hash: string (hash)
  - input: string (hex data)
  - isSystemTx: boolean
  - maxFeePerBlobGas: string (hex integer)
  - maxFeePerGas: string (hex integer)
  - maxPriorityFeePerGas: string (hex integer)
  - mint: string (hex integer)
  - nonce: string (hex integer)
  - r: string (hex integer)
  - s: string (hex integer)
  - sourceHash: string (hash)
  - to: string (address)
  - transactionIndex: string (hex integer)
  - type: integer
  - v: string (hex integer)
  - value: string (hex integer)
  - yParity: string (hex integer)

</TabItem>
</Tabs>

txpoolinspect

Returns a detailed info on tx pool transactions.

<Tabs>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "txpoolinspect",
      "params": []
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: object
  - pending: map of map of string
  - queued: map of map of string

</TabItem>
</Tabs>

txpoolstatus

Returns a tx pool status.

<Tabs>
<TabItem value="request" label="Request" default>
```

```
bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "txpoolstatus",
      "params": []
    }'


</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}


result: object
  - pending: string (hex integer)
  - queued: string (hex integer)

</TabItem>
</Tabs>



# web3.md:

---
title: web3 namespace
sidebarlabel: web3
sidebarposition: 10
---

import Tabs from "@theme/Tabs";
import TabItem from "@theme/TabItem";

web3clientVersion

Returns the current client version.

<Tabs>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "web3clientVersion",
      "params": []
    }'


</TabItem>
```

<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}

result: string

</TabItem>
</Tabs>

web3sha3

Returns Keccak of the given data.

<Tabs>
<TabItem value="params" label="Parameters">

1. data: string (hex data)

</TabItem>
<TabItem value="request" label="Request" default>

bash
curl localhost:8545 \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{
      "jsonrpc": "2.0",
      "id": 0,
      "method": "web3sha3",
      "params": [data]
    }'

</TabItem>
<TabItem value="response" label="Response">

json
{
  "jsonrpc": "2.0",
  "id": 0,
  "result": result
}

result: string (hash)

</TabItem>
</Tabs>

health-check.md:

---
title: Health check
sidebarposition: 1

:::warning
This article requires a revision.
:::

import Tabs from "@theme/Tabs";
import TabItem from "@theme/TabItem";

Nethermind has a pre-packed Nethermind.HealthChecks.dll plugin that allows you to monitor your Nethermind node better.
It leverages the power
of AspNetCore.Diagnostics.HealthChecks. It simply adds
an/healthendpoint to the JSON RPC service which can be used to check the Nethermind's liveness - verify if the
node is synced and has at least one peer. Useful when you don't want to query the node before it's able to
provide you data available only for fully synced nodes like ethgetBalance.

The Nethermind.HealthChecks.dllplugin will be automatically loaded on Nethermind start.

Enabling and configuring Health Checks

The health checks need to be additionally enabled which can be done either through --HealthChecks. flags or by adding
a "HealthChecks" section to the config file.

```
 json title="HealthChecks config section example"
  "HealthChecks": {
    "Enabled": true,
    "WebhooksEnabled": true,
    "WebhooksUri": "https://slack.webhook",
    "UIEnabled": true,
    "PollingInterval": 10,
    "Slug": "/api/health"
  }
```

:::danger
JSON RPC Service needs to be enabled in order for health checks to work --JsonRpc.Enabled true
:::

Each configuration option is described here.

Enabling Health Checks without UI

bash
```
./Nethermind.Runner --HealthChecks.Enabled true
```

The health endpoint is now available at localhost:8545/health by default (if your --JsonRpc.Port is 8545).
The health endpoint can be configured via --HealthChecks.Slug parameter e.g. --HealthChecks.Slug /api/health. We
can if it is working with curl:

bash
```
// Request
curl localhost:8545/health

// Example of response for Unhealthy node
{"status":"Unhealthy","totalDuration":"00:00:00.0015582","entries":{"node-health":{"data":{},"description":"The node has 0 peers
```

connected","duration":"00:00:00.0003881","status":"Unhealthy","tags":[]}}}

// Example of response for Healthy node
{"status":"Healthy","totalDuration":"00:00:00.0015582","entries":{"node-health":{"data":{},"description":"The node is now fully synced with a network, number of peers: 99","duration":"00:00:00.0003881","status":"Healthy","tags":[]}}}

:::info

Unhealthy returns 503 (Service Unavailable) status code
:::

:::info
Healthy returns 200 status code
:::

Enabling Health Checks UI

bash
./Nethermind.Runner --HealthChecks.Enabled true --HealthChecks.UIEnabled true

Enabling UI will expose an additional endpoint /healthchecks-uiand will allow seeing node's health on a nice UI. To
view the UI simply go to http://localhost:8545/healthchecks-ui.

!Unhealthy status reported on UI page.png>)

Enabling Slack reports

We may also add Slack Webhook endpoint to which our node's health will be reported. We need to pass
the --HealthChecks.WebhooksEnabled true and add the --HealthChecks.WebhooksUri which can be found in your Slack app
configuration.

bash
nethermind --HealthChecks.Enabled true --HealthChecks.UIEnabled true --HealthChecks.WebhooksEnabled true --HealthChecks.WebhooksUri https://hooks.slack.com/

If your node will be Unhealthy you should receive a message similar to this:

!Unhealthy

with description of why the node is unhealthy, node's name and information about the machine on which the node is
running.\
When it becomes Healthy (synced and with peers) you should receive:

!Healthy.png>)

Consensus Client health

This check verifies if the client receives messages from the CL. If you see this warning in your logs, it means that
there is something wrong with CL/Nethermind communication. Check more about setting up Nethermind and
CL here.

No incoming messages from Consensus Client. Consensus Client is required to sync

the node. Please make sure that it's working properly.

:::warning
Note that Consensus Client is required for normal node operations.
:::

health\nodeStatus

Health checks via JSON RPC requests were introduced in version v.1.10.18. For that, HealthChecks.Enabled should be
set to true.

<Tabs>
<TabItem value="request" label="Request">

```
{ "jsonrpc":"2.0","method":"healthnodeStatus","params":[],"id":67 }
```

</TabItem>
<TabItem label="Response" value="response">


```
{
    "jsonrpc": "2.0",
    "result": {
        "healthy": false,
        "messages": [
            "Sync degraded",
            "No messages from CL"
        ],
        "errors": [
            "SyncDegraded",
            "ClUnavailable"
        ],
        "isSyncing": true
    },
    "id": 67
}
```


</TabItem>
</Tabs>

Monitoring available storage space

Feature which is helping to track free disk space is enabled by default and monitors a drive which has been used to
configure database location. There are two new configuration options available:

 --HealthChecks.LowStorageSpaceWarningThreshold - Percentage of free disk space below which a warning will be
  displayed. If Health Checks UI is enabled, it will also be reported under node's health. Default value is 5 - meaning
  5% of free disk space.
 --HealthChecks.LowStorageSpaceShutdownThreshold - Percentage of available disk space below which node will shutdown
  to avoid database corruption. Default value is 1 - meaning 1% of free disk space.

bash
nethermind --HealthChecks.LowStorageSpaceWarningThreshold 5 --
HealthChecks.LowStorageSpaceShutdownThreshold 1

HealthChecks for producing and processing blocks

There are two fields for HealthChecks config: MaxIntervalWithoutProcessedBlock
and MaxIntervalWithoutProducedBlock. The
node will return unhealthy status if the interval elapsed without processing or
producing a block. Let's use the below
config as an example. If the node doesn't process a block for 15 seconds, we
will return unhealthy status. Analogically,
we will be waiting 45 seconds for a newly produced block.

 json title="HealthChecks config section example"

  "HealthChecks": {
    "Enabled": true,
    "WebhooksEnabled": true,
    "UIEnabled": true,
    "Slug": "/api/health",
    "MaxIntervalWithoutProcessedBlock ": 15,
    "MaxIntervalWithoutProducedBlock": 45
  }

If those fields are not set in a config, application will try to use them based
on seal engine specification. If there
is infinite time, unhealthy status can still be reported if processing or
producing threads stopped.

dotnet-counters.md:

---
title: dotnet-counters
sidebarposition: 1

This guide will walk you through setting up performance counters using the
dotnet-counters performance monitoring tool that observes counters published via
the EventCounters API.

Step 1: Install dotnet-counters

dotnet-counters can be either installed locally or in a Docker container.

Installing locally

Use the dotnet tool install command as follows:

bash
dotnet tool install -g dotnet-counters

Once installed, you can run the tool from the command line by typing dotnet-
counters.

Installing in a Docker container

To install dotnet-counters in a Docker container, create a Dockerfile with the
following content:

docker title="Dockerfile"
```
FROM mcr.microsoft.com/dotnet/sdk:8.0

RUN dotnet tool install -g dotnet-counters

ENV PATH="$PATH:/root/.dotnet/tools"

ENTRYPOINT ["/bin/bash"]
```

Then, build the Docker image:

bash
```
docker build -t dotnet-counters .
```

Step 2: Run Nethermind

To enable performance counters in Nethermind, use the command line option --Metrics.CountersEnabled true. For more options, see the Metrics configuration section.

:::warning Important
A consensus client of your choice must be running before you start Nethermind.
:::

Running locally

To enable performance counters, run Nethermind as follows:

bash
```
nethermind -c mainnet --Metrics.CountersEnabled true
```

Running in a Docker container

The easiest way of collecting metrics in a Docker container is to use Docker Compose. Below, we use the Nethermind official Docker image and the dotnet-counters image we created earlier:

yaml title="docker-compose.yml"
```
services:

  dotnet-counters:
    image: dotnet-counters
    containername: dotnet-counters
    stdinopen: true
    tty: true
    pid: service:nethermind
    volumes:
      - metrics:/tmp
    dependson:
      - nethermind

  nethermind:
    image: nethermind/nethermind:latest
    containername: nethermind
    restart: unless-stopped
    ports:
      - 8545:8545
      - 8551:8551
      - 30303:30303
    command: -c mainnet --Metrics.CountersEnabled true
```

```
    volumes:
      - ./keystore:/nethermind/keystore
      - ./logs:/nethermind/logs
      - ./netherminddb:/nethermind/netherminddb
      - metrics:/tmp

volumes:
  metrics:
```

:::info
dotnet-counters uses IPC socket communication to monitor the target process. For this, we use the metrics volume to share the IPC socket directory with the nethermind and dotnet-counter services. The pid option in the dotnet-counters service is used to share the PID namespace with the nethermind service. This is necessary for dotnet-counters to be able to see the Nethermind process.
:::

We can run the above file as follows:

```
bash
docker compose up
```

Step 3: Collect metrics

Once dotnet-counters is installed and Nethermind is running, we can start collecting the metrics. If you chose to collect metrics in the containers, run the following command in the dotnet-counters container:

```
bash
dotnet-counters collect -n nethermind
```

By default, dotnet-counters stores the collected metrics in the current directory in CSV format. However, you may also store them in JSON format and another directory. For instance:

```
bash
dotnet-counters collect -n nethermind -f json -o /tmp/counters.json
```

For more info about dotnet-counters, see its official docs.

grafana-and-prometheus.md:

```
---
title: Grafana and Prometheus
sidebarposition: 0
---
```

This guide will walk you through setting up a local metrics infrastructure using Grafana and Prometheus.

Step 1: Set up Grafana and Prometheus

To simplify the process, we will use the metrics-infrastructure repository, which contains the necessary configuration files to run Grafana and Prometheus in a Docker container. However, you can also set up Grafana and Prometheus manually the way it fits your specific needs.

```
bash
git clone https://github.com/NethermindEth/metrics-infrastructure.git
```

Step 2: Run the stack

After cloning the repository, navigate to its root directory and run the containers using Docker Compose as follows:

bash
docker compose up

Once the stack is running, you can access the following services:

- Grafana: localhost:3000\
  Use admin for both the username and password. When asked for a password change, you may skip it. Then, navigate to Dashboards > Nethermind Dashboard.
- Prometheus: localhost:9090
- Pushgateway: localhost:9091\
  To specify another endpoint for the Pushgateway, use the --Metrics.PushGatewayUrl command line option.

Step 3: Run Nethermind

To enable metrics in Nethermind, use the --Metrics.Enabled true command line option. For more options, see the Metrics configuration section.

:::warning Important
A consensus client of your choice must be running before you start Nethermind.
:::

Run Nethermind as follows:

bash
nethermind -c mainnet --Metrics.Enabled true

Alternatively, you may add the nethermind service to the docker-compose.yml file in the repository root to run everything altogether:

yaml title="docker-compose.yml"
```
nethermind:
  image: nethermind/nethermind:latest
  containername: nethermind
  restart: unless-stopped
  ports:
    - 8545:8545
    - 8551:8551
    - 30303:30303
  ulimits:
    nofile:
      soft: 1000000
      hard: 1000000
  command: -c mainnet --Metrics.Enabled true
  volumes:
    - ./keystore:/nethermind/keystore
    - ./logs:/nethermind/logs
    - ./netherminddb:/nethermind/netherminddb
```

In this case, you may want to configure your consensus client similarly.

metrics.md:

---

title: Metrics
sidebarposition: 0
tocmaxheadinglevel: 4

Currently, Nethermind provides the following options to monitor and collect metrics about itself:

- Grafana and Prometheus
- dotnet-counters

Parameters by namespace

<!--[start autogen]-->

<details>
<summary className="nd-details-heading">

Aura

</summary>

- nethermindaurastep

  Current AuRa step

- nethermindcommithashtransaction

  RANDAO number of commit hash transactions

- nethermindemitinitiatechange

  POSDAO number of emit init change transactions

- nethermindreportedbenignmisbehaviour

  Number of reported benign misbehaviour validators

- nethermindreportedmaliciousmisbehaviour

  Number of reported malicious misbehaviour validators

- nethermindrevealnumber

  RANDAO number of reveal number transactions

- nethermindsealedtransactions

  Number of sealed transactions generated by engine

- nethermindvalidatorscount

  Number of current AuRa validators

</details>

<details>
<summary className="nd-details-heading">

Blockchain

</summary>
<p>

- nethermindbestknownblocknumber

  The estimated highest block available.

- nethermindblockchainheight

  The current height of the canonical chain.

- nethermindblocks

  Total number of blocks processed

- nethermindblockssealed

  Total number of sealed blocks

- nethermindfailedblockseals

  Total number of failed block seals

- nethermindgaslimit

  Gas Limit for processed blocks

- nethermindgasused

  Gas Used in processed blocks

- nethermindlastdifficulty

  Difficulty of the last block

- nethermindmgas

  Total MGas processed

- nethermindmgaspersec

  MGas processed per second

- nethermindprocessingqueuesize

  Number of blocks awaiting for processing.

- nethermindrecoveryqueuesize

  Number of blocks awaiting for recovery of public keys from signatures.

- nethermindreorganizations

  Total number of chain reorganizations

- nethermindtotaldifficulty

  Total difficulty on the chain

- nethermindtransactions

  Total number of transactions processed

</details>

<details>
<summary className="nd-details-heading">

Db

</summary>

- nethermindcodedbcache

  Number of Code DB cache reads.

- netherminddbblockcachesize

  Database block cache size per database

- netherminddbcompactionstats

  Metrics extracted from RocksDB Compaction Stats

- netherminddbindexfiltersize

  Database index and filter size per database

- netherminddbmemtablesize

  Database memtable per database

- netherminddbreads

  Database reads per database

- netherminddbsize

  Database size per database

- netherminddbstats

  Metrics extracted from RocksDB Compaction Stats and DB Statistics

- netherminddbwrites

  Database writes per database

- nethermindstatedbpruning

  Indicator if StadeDb is being pruned.

- nethermindstatereaderreads

  Number of State Reader reads.

- nethermindstatetreecache

  Number of State Trie cache hits.

- nethermindstatetreereads

  Number of State Trie reads.

- nethermindstatetreewrites

  Number of Blocks Trie writes.

- nethermindstoragereaderreads

  Number of storage reader reads.

- nethermindstoragetreecache

  Number of storage trie cache hits.

- nethermindstoragetreereads

  Number of storage trie reads.

- nethermindstoragetreewrites

  Number of storage trie writes.

- nethermindthreadlocalcodedbcache

  Number of Code DB cache reads on thread.

- nethermindthreadlocalstatetreereads

  Number of State Trie reads on thread.

- nethermindthreadlocalstoragetreereads

  Number of storage trie reads on thread.

</details>

<details>
<summary className="nd-details-heading">

Evm

</summary>

- nethermindblockhashopcode

  Number of BLOCKHASH opcodes executed.

- nethermindbn254addprecompile

  Number of BN254ADD precompile calls.

- nethermindbn254mulprecompile

  Number of BN254MUL precompile calls.

- nethermindbn254pairingprecompile

  Number of BN254PAIRING precompile calls.

- nethermindcalls

  Number of calls to other contracts.

- nethermindcontractsanalysed

  Number of contracts' code analysed for jump destinations.

- nethermindcreates

  Number of contract create calls.

- nethermindecrecoverprecompile

  Number of ECRECOVERY precompile calls.

- nethermindemptycalls

  Number of calls made to addresses without code.

- nethermindevmexceptions

  Number of EVM exceptions thrown by contracts.

- nethermindexpopcode

  Number of EXP opcodes executed.

- nethermindmcopyopcode

  Number of MCOPY opcodes executed.

- nethermindmodexpprecompile

  Number of MODEXP precompile calls.

- nethermindpointevaluationprecompile

  Number of Point Evaluation precompile calls.

- nethermindripemd160precompile

  Number of RIPEMD160 precompile calls.

- nethermindsecp256r1precompile

  Number of Secp256r1 precompile calls.

- nethermindselfdestructs

  Number of SELFDESTRUCT calls.

- nethermindsha256precompile

  Number of SHA256 precompile calls.

- nethermindsloadopcode

  Number of SLOAD opcodes executed.

- nethermindsstoreopcode

  Number of SSTORE opcodes executed.

- nethermindthreadlocalcalls

  Number of calls to other contracts on thread.

- nethermindthreadlocalcontractsanalysed

  Number of contracts' code analysed for jump destinations on thread.

- nethermindthreadlocalcreates

  Number of contract create calls on thread.

- nethermindthreadlocalemptycalls

  Number of calls made to addresses without code on thread.

- nethermindthreadlocalsloadopcode

  Number of SLOAD opcodes executed on thread.

- nethermindthreadlocalsstoreopcode

  Number of SSTORE opcodes executed on thread.

- nethermindtloadopcode

  Number of TLOAD opcodes executed.

- nethermindtstoreopcode

  Number of TSTORE opcodes executed.

</details>

<details>
<summary className="nd-details-heading">

JsonRpc

</summary>

- nethermindjsonrpcbytesreceived

  Number of JSON RPC bytes received.

- nethermindjsonrpcbytessent

  Number of JSON RPC bytes sent.

- nethermindjsonrpcerrors

  Number of JSON RPC requests processed with errors.

- nethermindjsonrpcinvalidrequests

  Number of JSON RPC requests that were invalid.

- nethermindjsonrpcrequestdeserializationfailures

  Number of JSON RPC requests that failed JSON deserialization.

- nethermindjsonrpcrequests

  Total number of JSON RPC requests received by the node.

- nethermindjsonrpcsuccesses

Number of JSON RPC requests processed successfully.

</details>

<details>
<summary className="nd-details-heading">

Merge

</summary>

- nethermindforkchoiceupdedexecutiontime

  ForkchoiceUpded request execution time

- nethermindgetpayloadrequests

  Number of GetPayload Requests

- nethermindnewpayloadexecutiontime

  NewPayload request execution time

- nethermindnumberoftransactionsingetpayload

  Number of Transactions included in the Last GetPayload Request

</details>

<details>
<summary className="nd-details-heading">

Network

</summary>

- nethermindhandshakes

  Number of devp2p handshakes

- nethermindhandshaketimeouts

  Number of devp2p handshake timeouts

- nethermindincomingconnections

  Number of incoming connection.

- nethermindincomingp2pmessagebytes

  Bytes of incoming p2p packets.

- nethermindincomingp2pmessages

  Number of incoming p2p packets.

- nethermindlocaldisconnectstotal

  Number of local disconnects

- nethermindoutgoingconnections

  Number of outgoing connection.

- nethermindoutgoingp2pmessagebytes

  Bytes of outgoing p2p packets.

- nethermindoutgoingp2pmessages

  Number of outgoing p2p packets.

- nethermindpeerlimit

  The maximum number of peers this node allows to connect.

- nethermindremotedisconnectstotal

  Number of remote disconnects

</details>

<details>
<summary className="nd-details-heading">

Pruning

</summary>

- nethermindcachednodescount

  Nodes that are currently kept in cache (either persisted or not)

- nethermindcommittednodescount

  Nodes that have been committed since the session start. These nodes may have been pruned, persisted or replaced.

- nethermindeepprunedpersistednodescount

  Nodes that have been removed from the cache during deep pruning because they have been persisted before.

- nethermindeeppruningtime

  Time taken by the last deep pruning.

- nethermindlastpersistedblocknumber

  Last persisted block number (snapshot).

- nethermindloadedfromcachenodescount

  Number of reads from the node cache.

- nethermindloadedfromdbnodescount

  Number of DB reads.

- nethermindloadedfromrlpcachenodescount

  Number of reads from the RLP cache.

- nethermindmemoryusedbycache

  Estimated memory used by cache.

- nethermindpersistednodecount

  Nodes that have been persisted since the session start.

- nethermindprunedpersistednodescount

  Nodes that have been removed from the cache during pruning because they have been persisted before.

- nethermindprunedtransientnodescount

  Nodes that have been removed from the cache during pruning because they were no longer needed.

- nethermindpruningtime

  Time taken by the last pruning.

- nethermindremovednodecount

  Nodes that was removed via live pruning.

- nethermindreplacednodescount

  Number of nodes that have been exactly the same as other nodes in the cache when committing.

- nethermindsnapshotpersistencetime

  Time taken by the last snapshot persistence.

</details>

<details>
<summary className="nd-details-heading">

Runner

</summary>

- nethermindversion

  Version number

</details>

<details>
<summary className="nd-details-heading">

Synchronization

</summary>

- nethermindsyncpeers

  Number of sync peers.

</details>

<details>
<summary className="nd-details-heading">

Trie

</summary>

- nethermindtreenodehashcalculations

  Number of trie node hash calculations.

- nethermindtreenoderlpdecodings

  Number of trie node RLP decodings.

- nethermindtreenoderlpencodings

  Number of trie node RLP encodings.

</details>

<details>
<summary className="nd-details-heading">

TxPool

</summary>

- nethermindblobsinblock

  Number of blobs in the block.

- nethermindblobtransactioncount

  Number of blob transactions in pool.

- nethermindblobtransactionsinblock

  Number of blob transactions in the block.

- netherminddarkpoolratiolevel1

  Ratio of transactions in the block absent in hashCache.

- netherminddarkpoolratiolevel2

  Ratio of transactions in the block absent in pending transactions.

- nethermindeip1559transactionsratio

  Ratio of 1559-type transactions in the block.

- nethermindpending1559transactionsadded

  Number of pending 1559-type transactions added to transaction pool.

- nethermindpendingblobtransactionsadded

  Number of pending blob-type transactions added to transaction pool.

- nethermindpendingtransactionsadded

  Number of pending transactions added to transaction pool.

- nethermindpendingtransactionsbalancebelowvalue

  Number of pending transactions received that were ignored because balance is less than txn value.

- nethermindpendingtransactionsconflictingtxtype

  Number of transactions rejected because of already pending tx of other type (allowed blob txs or others, not both at once).

- nethermindpendingtransactionsdiscarded

  Number of pending transactions received that were ignored.

- nethermindpendingtransactionsevicted

  Number of pending transactions evicted from transaction pool.

- nethermindpendingtransactionsgaslimittoohigh

  Number of pending transactions received that were ignored because the gas limit was to high for the block.

- nethermindpendingtransactionshashesreceived

  Number of hashes of pending transactions received from peers.

- nethermindpendingtransactionshashessent

  Number of hashes of pending transactions broadcasted to peers.

- nethermindpendingtransactionsknown

  Number of already known pending transactions.

- nethermindpendingtransactionslownonce

  Number of transactions with already used nonce.

- nethermindpendingtransactionsmalformed

  Number of malformed transactions.

- nethermindpendingtransactionsnoncegap

  Number of pending transactions received that were ignored because of not having preceding nonce of this sender in TxPool.

- nethermindpendingtransactionsnoncetoofarinfuture

  Number of transactions with nonce too far in future.

- nethermindpendingtransactionsnotsupportedtxtype

  Number of pending transactions received that were ignored because of not supported transaction type.

- nethermindpendingtransactionspassedfiltersbutcannotcompeteonfees

  Number of pending transactions received that were ignored after passing early rejections as balance is too low to compete with lowest effective fee in transaction pool.

- nethermindpendingtransactionspassedfiltersbutcannotreplace

  Number of pending transactions received that were trying to replace tx with the same sender and nonce and failed.

- nethermindpendingtransactionsreceived

  Number of pending transactions received from peers.

- nethermindpendingtransactionssent

  Number of pending transactions broadcasted to peers.

- nethermindpendingtransactionstoolowbalance

  Number of pending transactions received that were ignored because balance too low for fee to be higher than the lowest fee in transaction pool.

- nethermindpendingtransactionstoolowfee

  Number of pending transactions received that were ignored because of fee lower than the lowest fee in transaction pool.

- nethermindpendingtransactionstoolowpriorityfee

  Number of pending transactions received that were ignored because of priority fee lower than minimal requirement.

- nethermindpendingtransactionsunresolvablesender

  Number of pending transactions received that were ignored because the sender couldn't be resolved.

- nethermindpendingtransactionswithexpensivefiltering

  Number of pending transactions that reached filters which are resource expensive

- nethermindpendingtransactionszerobalance

  Number of pending transactions received that were ignored because balance is zero and cannot pay gas.

- nethermindtransactioncount

  Number of transactions in pool.

</p>

</details>

<!--[end autogen]-->

aura.md:

---
title: Aura-based validators
sidebarposition: 0

This guide will walk you through configuring an Aura-based validator with Nethermind in a Docker container using the Energy Web chain as an example.

:::info
Your machine's clock has to be synchronized. Otherwise, you might miss block sealing. By default, the block time is set to 5 seconds.
:::

Configuring a Docker container

The example below shows how to configure a Docker container for an Aura-based validator on the Energy Web chain:

yaml title="docker-compose.yml"
services:
  nethermind-validator:
    image: nethermind/nethermind:latest
    containername: nethermind-validator
    restart: unless-stopped
    ports:
      - 8545:8545
      - 30303:30303
    ulimits:
      nofile:
        soft: 1000000
        hard: 1000000
    environment:
      - NETHERMINDCONFIG=energyweb
    volumes:
      - ./keystore:/nethermind/keystore
      - ./logs:/nethermind/logs
      - ./netherminddb:/nethermind/netherminddb

Configuring keyfile

Make sure that the keyfile name contains the public key (address). Otherwise, Nethermind doesn't recognize it as such.
For instance, a keyfile can be named key-0x1234567890123456789012345678901234567890.

The keyfile must be stored in the keystore directory located in the Nethermind base data directory.

Configuration settings

Here is an example of recommended settings for a validator. The most convenient way to configure these settings is either defining them in the configuration file or passing them as environment variables.

- Init.IsMining: true
- Init.MemoryHint: Can be left unspecified. It's recommended to configure it

accordingly to the machine specification(for Eneergy Web, 768000000 is enough).
- EthStats namespace parameters if you want to report node status to Ethstats for your network.
- Metrics namespace parameters to enable node monitoring.
- KeyStore.PasswordFiles: The path to the file containing the password for the mining private key.
- KeyStore.UnlockAccounts: An array of accounts. Provide the miner public address here.
- KeyStore.BlockAuthorAccount: The miner public address should be provided here as well.
- Aura.ForceSealing: true
- Merge.Enabled: false

Here's an example of above settings in the Energy Web configuration file:

json title="energyweb.cfg"

```
{
  "Init": {
    "ChainSpecPath": "chainspec/energyweb.json",
    "GenesisHash":
"0x0b6d3e680af2fc525392c720666cce58e3d8e6fe75ba4b48cb36bcc69039229b",
    "BaseDbPath": "netherminddb/energyweb",
    "LogFileName": "energyweb.log",
    "MemoryHint": 768000000
  },
  "Sync": {
    "FastSync": true,
    "PivotNumber": 26940000,
    "PivotHash":
"0x8835983de9578a4355313afd2a43d8eada6f2a4ddbd9c51da103e0d5f53c4d8b",
    "PivotTotalDifficulty": "916720696485008220570331192421183561625789827 4",
    "FastBlocks": true,
    "UseGethLimitsInFastBlocks": false,
    "FastSyncCatchUpHeightDelta": 10000000000
  },
  "EthStats": {
    "Enabled": true,
    "Name": "Nethermind Energy Web",
    "Secret": "secret...",
    "Url": "ws://localhost:3000/api"
  },
  "Metrics": {
    "Enabled": true,
    "NodeName": "Energy Web",
    "PushGatewayUrl": "http://localhost:9091/metrics"
  },
  "Mining": {
    "MinGasPrice": 1
  },
  "Merge": {
    "Enabled": false
  },
  "Aura": {
    "ForceSealing": true
  },
  "KeyStore": {
    "PasswordFiles": ["keystore/password"],
    "UnlockAccounts": ["0x..."],
    "BlockAuthorAccount": "0x..."
  },
}
```

Running the validator

The above Docker Compose file can be run from the directory the docker-compose.yml is located in as follows:

bash
docker compose up -d

To check the logs and verify the sealing of blocks, run:

bash
docker compose logs -f nethermind-validator

validators.md:

---
title: Validators
sidebarposition: 0

General considerations

:::warning Important
Please check out the security considerations before using Nethermind as a validator.
:::

For Ethereum validators, we highly recommend checking out Staking with Ethereum and Validator checklist.

Hardware configurations

The following hardware configurations for Ethereum Mainnet validators have been battle-tested by us and our users. We have observed excellent validator performance and stability with these configurations.

:::note
Before setting up your infrastructure, check out Nethermind hardware requirements.
:::

On-premises

A single validator on Intel NUC 11:

- CPU: Intel Core i7-1165G7
- Memory: Crucial 32GB DDR4-3200 SODIMM
- Storage: Samsung 980 PRO PCIe NVMe SSD 2TB
- Internet speed: 620 Mbps download, 160 Mbps upload

AWS

Multiple validators on the following EC2 instances:

- m6i.2xlarge: 8 vCPU, 32 GiB memory
- m7g.2xlarge: 8 vCPU, 32 GiB memory

These configurations have proven to work well for 1000-1500 validators and haven't been tested for more validators. Also, the validator clients have been separated from the consensus and execution clients and running on t4g.small instances.

Azure

Multiple validators on the following VM instances:

- StandardD8v5: 8 vCPU, 32 GiB memory
- StandardD8psv5: 8 vCPU, 32 GiB memory

These configurations have proven to work well for 1000-1500 validators and haven't been tested for more validators. Also, the validator clients have been separated from the consensus and execution clients and running on StandardD2plsv5 instances.

GCP

Multiple validators on the c2d-highmem-4 instance: 4 vCPU, 32 GB memory

These configurations have proven to work well for 1000-1500 validators and haven't been tested for more validators. Also, the validator clients have been separated from the consensus and execution clients and running on e2-small instances.

Gnosis validators

To set up a Gnosis Chain validator, see the Gnosis Chain documentation.