The Kubo config file

The Kubo (go-ipfs) config file is a JSON document located at `$IPFS_PATH/config`. It is read once at node instantiation, either for an offline command, or when starting the daemon. Commands that execute on a running daemon do not read the config file at runtime.

Table of Contents

Profiles

Configuration profiles allow to tweak configuration quickly. Profiles can be applied with the `--profile` flag to `ipfs init` or with the `ipfs config profile apply` command. When a profile is applied a backup of the configuration file will be created in `$IPFS_PATH`.

The available configuration profiles are listed below. You can also find them documented in `ipfs config profile --help`.

- `server`

  Disables local host discovery, recommended when running IPFS on machines with public IPv4 addresses.

- `randomports`

  Use a random port number for the incoming swarm connections.

- `default-datastore`

  Configures the node to use the default datastore (flatfs).

  Read the "flatfs" profile description for more information on this datastore.

  This profile may only be applied when first initializing the node.

- `local-discovery`

  Enables local discovery (enabled by default). Useful to re-enable local discovery after it's
  disabled by another profile (e.g., the server profile).

- `test`

  Reduces external interference of IPFS daemon, this
  is useful when using the daemon in test environments.

- `default-networking`

  Restores default network settings.
  Inverse profile of the test profile.

- `flatfs`

  Configures the node to use the flatfs datastore. Flatfs is the default datastore.

  This is the most battle-tested and reliable datastore.
  You should use this datastore if:

  - You need a very simple and very reliable datastore, and you trust your
    filesystem. This datastore stores each block as a separate file in the
    underlying filesystem so it's unlikely to lose data unless there's an issue
    with the underlying file system.
  - You need to run garbage collection in a way that reclaims free space as soon as possible.
  - You want to minimize memory usage.
  - You are ok with the default speed of data import, or prefer to use `--nocopy`.

  This profile may only be applied when first initializing the node.

- `badgerds`

  Configures the node to use the experimental badger datastore. Keep in mind that this **uses an outdated badger 1.x**.

  Use this datastore if some aspects of performance,
  especially the speed of adding many gigabytes of files, are critical. However, be aware that:

  - This datastore will not properly reclaim space when your datastore is
    smaller than several gigabytes. If you run IPFS with `--enable-gc`, you plan on storing very little data in
    your IPFS node, and disk usage is more critical than performance, consider using
    `flatfs`.

  - This datastore uses up to several gigabytes of memory.
  - Good for medium-size datastores, but may run into performance issues if your dataset is bigger than a terabyte.
  - The current implementation is based on old badger 1.x which is no longer supported by the upstream team.

  This profile may only be applied when first initializing the node.

- `lowpower`

  Reduces daemon overhead on the system. Affects node
  functionality - performance of content discovery and data
  fetching may be degraded. Local data won't be announced on routing systems like DHT.

  - `Swarm.ConnMgr` set to maintain minimum number of p2p connections at a time.
  - Disables `Reprovider` service → no CID will be announced on DHT and other routing systems(!)
  - Disables AutoNAT.

  Use this profile with caution.

Types

This document refers to the standard JSON types (e.g., `null`, `string`,
`number`, etc.), as well as a few custom types, described below.

`flag`

Flags allow enabling and disabling features. However, unlike simple booleans,
they can also be `null` (or omitted) to indicate that the default value should
be chosen. This makes it easier for Kubo to change the defaults in the
future unless the user *explicitly* sets the flag to either `true` (enabled) or
`false` (disabled). Flags have three possible states:

- `null` or missing (apply the default value).
- `true` (enabled)
- `false` (disabled)

`priority`

Priorities allow specifying the priority of a feature/protocol and disabling the
feature/protocol. Priorities can take one of the following values:

- `null`/missing (apply the default priority, same as with flags)
- `false` (disabled)
- `1 - 2^63` (priority, lower is preferred)

`strings`

Strings is a special type for conveniently specifying a single string, an array
of strings, or null:

- `null`
- `"a single string"`
- `["an", "array", "of", "strings"]`

`duration`

Duration is a type for describing lengths of time, using the same format go
does (e.g, `"1d2h4m40.01s"`).

`optionalInteger`

Optional integers allow specifying some numerical value which has
an implicit default when missing from the config file:

- `null`/missing will apply the default value defined in Kubo sources (`.WithDefault(value)`)
- an integer between `-2^63` and `2^63-1` (i.e. `-9223372036854775808` to `9223372036854775807`)

`optionalBytes`

Optional Bytes allow specifying some number of bytes which has
an implicit default when missing from the config file:

- `null`/missing (apply the default value defined in Kubo sources)
- a string value indicating the number of bytes, including human readable representations:
  - SI sizes (metric units, powers of 1000), e.g. `1B`, `2kB`, `3MB`, `4GB`, `5TB`, …)
  - IEC sizes (binary units, powers of 1024), e.g. `1B`, `2KiB`, `3MiB`, `4GiB`, `5TiB`, …)

`optionalString`

Optional strings allow specifying some string value which has
an implicit default when missing from the config file:

- `null`/missing will apply the default value defined in Kubo sources (`.WithDefault("value")`)
- a string

`optionalDuration`

Optional durations allow specifying some duration value which has
an implicit default when missing from the config file:

- `null`/missing will apply the default value defined in Kubo sources (`.WithDefault("1h2m3s")`)
- a string with a valid go duration (e.g, `"1d2h4m40.01s"`).

`Addresses`

Contains information about various listener addresses to be used by this node.

`Addresses.API`

Multiaddr or array of multiaddrs describing the address to serve the local HTTP
API on.

Supported Transports:

- tcp/ip{4,6} - `/ipN/.../tcp/...`
- unix - `/unix/path/to/socket`

Default: `/ip4/127.0.0.1/tcp/5001`

Type: `strings` (multiaddrs)

`Addresses.Gateway`

Multiaddr or array of multiaddrs describing the address to serve the local gateway on.

Supported Transports:

- tcp/ip{4,6} - `/ipN/.../tcp/...`
- unix - `/unix/path/to/socket`

Default: `/ip4/127.0.0.1/tcp/8080`

Type: `strings` (multiaddrs)

`Addresses.Swarm`

An array of multiaddrs describing which addresses to listen on for p2p swarm connections.

Supported Transports:

- tcp/ip{4,6} - `/ipN/.../tcp/...`
- websocket - `/ipN/.../tcp/.../ws`
- quic (Draft-29) - `/ipN/.../udp/.../quic` - can share the same two tuple with `/quic-v1` and `/quic-v1/webtransport`
- quicv1 (RFC9000) - `/ipN/.../udp/.../quic-v1` - can share the same two tuple with `/quic` and `/quic-v1/webtransport`
- webtransport `/ipN/.../udp/.../quic-v1/webtransport` - can share the same two tuple with `/quic` and `/quic-v1`

Default:

```
[
  "/ip4/0.0.0.0/tcp/4001",
  "/ip6/::/tcp/4001",
  "/ip4/0.0.0.0/udp/4001/quic",
  "/ip4/0.0.0.0/udp/4001/quic-v1",
  "/ip4/0.0.0.0/udp/4001/quic-v1/webtransport",
  "/ip6/::/udp/4001/quic",
  "/ip6/::/udp/4001/quic-v1",
  "/ip6/::/udp/4001/quic-v1/webtransport"
]
```

Type: `array[string]` (multiaddrs)

`Addresses.Announce`

If non-empty, this array specifies the swarm addresses to announce to the network. If empty, the daemon will announce inferred swarm addresses.

Default: `[]`

Type: `array[string]` (multiaddrs)

`Addresses.AppendAnnounce`

Similar to `Addresses.Announce` except this doesn't override inferred swarm addresses if non-empty.

Default: `[]`

Type: `array[string]` (multiaddrs)

`Addresses.NoAnnounce`

An array of swarm addresses not to announce to the network.
Takes precedence over `Addresses.Announce` and `Addresses.AppendAnnounce`.

Default: `[]`

Type: `array[string]` (multiaddrs)

`API`

Contains information used by the API gateway.

`API.HTTPHeaders`

Map of HTTP headers to set on responses from the API HTTP server.

Example:

```
{
    "Foo": ["bar"]
}
```

Default: `null`

Type: `object[string -> array[string]]` (header names -> array of header values)

`AutoNAT`

Contains the configuration options for the AutoNAT service. The AutoNAT service helps other nodes on the network determine if they're publicly reachable from the rest of the internet.

`AutoNAT.ServiceMode`

When unset (default), the AutoNAT service defaults to *enabled*. Otherwise, this field can take one of two values:

- "enabled" - Enable the service (unless the node determines that it, itself, isn't reachable by the public internet).
- "disabled" - Disable the service.

Additional modes may be added in the future.

Type: `string` (one of `"enabled"` or `"disabled"`)

`AutoNAT.Throttle`

When set, this option configures the AutoNAT services throttling behavior. By default, Kubo will rate-limit the number of NAT checks performed for other nodes to 30 per minute, and 3 per peer.

`AutoNAT.Throttle.GlobalLimit`

Configures how many AutoNAT requests to service per `AutoNAT.Throttle.Interval`.

Default: 30

Type: `integer` (non-negative, `0` means unlimited)

`AutoNAT.Throttle.PeerLimit`

Configures how many AutoNAT requests per-peer to service per `AutoNAT.Throttle.Interval`.

Default: 3

Type: `integer` (non-negative, `0` means unlimited)

`AutoNAT.Throttle.Interval`

Configures the interval for the above limits.

Default: 1 Minute

Type: `duration` (when `0`/unset, the default value is used)

`Bootstrap`

Bootstrap is an array of multiaddrs of trusted nodes that your node connects to, to fetch other nodes of the network on startup.

Default: The ipfs.io bootstrap nodes

Type: `array[string]` (multiaddrs)

`Datastore`

Contains information related to the construction and operation of the on-disk storage system.

`Datastore.StorageMax`

A soft upper limit for the size of the ipfs repository's datastore. With `StorageGCWatermark`, is used to calculate whether to trigger a gc run (only if `--enable-gc` flag is set).

Default: `"10GB"`

Type: `string` (size)

`Datastore.StorageGCWatermark`

The percentage of the `StorageMax` value at which a garbage collection will be triggered automatically if the daemon was run with automatic gc enabled (that option defaults to false currently).

Default: `90`

Type: `integer` (0-100%)

`Datastore.GCPeriod`

A time duration specifying how frequently to run a garbage collection. Only used if automatic gc is enabled.

Default: `1h`

Type: `duration` (an empty string means the default value)

`Datastore.HashOnRead`

A boolean value. If set to true, all block reads from the disk will be hashed and verified. This will cause increased CPU utilization.

Default: `false`

Type: `bool`

`Datastore.BloomFilterSize`

A number representing the size in bytes of the blockstore's bloom filter. A value of zero represents
the feature is disabled.

This site generates useful graphs for various bloom filter values:
https://hur.st/bloomfilter/?n=1e6&p=0.01&m=&k=7 You may use it to find a

preferred optimal value, where `m` is `BloomFilterSize` in bits. Remember to convert the value `m` from bits, into bytes for use as `BloomFilterSize` in the config file. For example, for 1,000,000 blocks, expecting a 1% false-positive rate, you'd end up with a filter size of 9592955 bits, so for `BloomFilterSize` we'd want to use 1199120 bytes. As of writing, 7 hash functions
are used, so the constant `k` is 7 in the formula.

Default: `0` (disabled)

Type: `integer` (non-negative, bytes)

`Datastore.Spec`

Spec defines the structure of the ipfs datastore. It is a composable structure, where each datastore is represented by a json object. Datastores can wrap other datastores to provide extra functionality (eg metrics, logging, or caching).

This can be changed manually, however, if you make any changes that require a different on-disk structure, you will need to run the ipfs-ds-convert tool to migrate data into the new
structures.

For more information on possible values for this configuration option, see docs/datastores.md

Default:

```
{
  "mounts": [
    {
      "child": {
        "path": "blocks",
        "shardFunc": "/repo/flatfs/shard/v1/next-to-last/2",
        "sync": true,
        "type": "flatfs"
      },
      "mountpoint": "/blocks",
      "prefix": "flatfs.datastore",
      "type": "measure"
    },
    {
      "child": {
        "compression": "none",
        "path": "datastore",
        "type": "levelds"
      },
      "mountpoint": "/",
      "prefix": "leveldb.datastore",
      "type": "measure"
    }
  ],
  "type": "mount"
}
```

Type: `object`

`Discovery`

Contains options for configuring IPFS node discovery mechanisms.

`Discovery.MDNS`

Options for ZeroConf Multicast DNS-SD peer discovery.

`Discovery.MDNS.Enabled`

A boolean value for whether or not Multicast DNS-SD should be active.

Default: `true`

Type: `bool`

`Discovery.MDNS.Interval`

REMOVED: this is not configurable anymore
in the new mDNS implementation.

`Experimental`

Toggle and configure experimental features of Kubo. Experimental features are listed here.

`Gateway`

Options for the HTTP gateway.

`Gateway.NoFetch`

When set to true, the gateway will only serve content already in the local repo
and will not fetch files from the network.

Default: `false`

Type: `bool`

`Gateway.NoDNSLink`

A boolean to configure whether DNSLink lookup for value in `Host` HTTP header
should be performed. If DNSLink is present, the content path stored in the DNS TXT
record becomes the `/` and the respective payload is returned to the client.

Default: `false`

Type: `bool`

`Gateway.HTTPHeaders`

Headers to set on gateway responses.

Default:

```json
{
    "Access-Control-Allow-Headers": [
        "X-Requested-With"
    ],
    "Access-Control-Allow-Methods": [
        "GET"
    ],
    "Access-Control-Allow-Origin": [
        "*"
    ]
}
```

Type: `object[string -> array[string]]`

`Gateway.RootRedirect`

A url to redirect requests for `/` to.

Default: `""`

Type: `string` (url)

`Gateway.FastDirIndexThreshold`

REMOVED: this option is no longer necessary. Ignored since Kubo 0.18.

`Gateway.Writable`

REMOVED: this option no longer available as of Kubo 0.20.

We are working on developing a modern replacement. To support our efforts, please leave a comment describing your use case in ipfs/specs#375.

`Gateway.PathPrefixes`

REMOVED: see go-ipfs#7702

`Gateway.PublicGateways`

`PublicGateways` is a dictionary for defining gateway behavior on specified hostnames.

Hostnames can optionally be defined with one or more wildcards.

Examples:

- `*.example.com` will match requests to `http://foo.example.com/ipfs/*` or `http://{cid}.ipfs.bar.example.com/*`.
- `foo-*.example.com` will match requests to `http://foo-bar.example.com/ipfs/*` or `http://{cid}.ipfs.foo-xyz.example.com/*`.

`Gateway.PublicGateways: Paths`

An array of paths that should be exposed on the hostname.

Example:

```
{
  "Gateway": {
    "PublicGateways": {
      "example.com": {
        "Paths": ["/ipfs", "/ipns"],
      }
    }
  }
}
```

Above enables `http://example.com/ipfs/*` and `http://example.com/ipns/*` but not `http://example.com/api/*`

Default: `[]`

Type: `array[string]`

`Gateway.PublicGateways: UseSubdomains`

A boolean to configure whether the gateway at the hostname provides Origin isolation between content roots.

- `true` - enables subdomain gateway at `http://*.{hostname}/`
  - **Requires whitelist:** make sure respective `Paths` are set.
    For example, `Paths: ["/ipfs", "/ipns"]` are required for `http://{cid}.ipfs.{hostname}` and `http://{foo}.ipns.{hostname}` to work:

    ```
    "Gateway": {
        "PublicGateways": {
            "dweb.link": {
                "UseSubdomains": true,
                "Paths": ["/ipfs", "/ipns"],
            }
        }
    }
    ```

    - **Backward-compatible:** requests for content paths such as `http://{hostname}/ipfs/{cid}` produce redirect to `http://{cid}.ipfs.{hostname}`
    - **API:** if `/api` is on the `Paths` whitelist, `http://{hostname}/api/{cmd}` produces redirect to `http://api.{hostname}/api/{cmd}`

- `false` - enables path gateway at `http://{hostname}/*`

    - Example:

      ```
      "Gateway": {
          "PublicGateways": {
              "ipfs.io": {
                  "UseSubdomains": false,
                  "Paths": ["/ipfs", "/ipns", "/api"],
              }
          }
      }
      ```

Default: `false`

Type: `bool`

`Gateway.PublicGateways: NoDNSLink`

A boolean to configure whether DNSLink for hostname present in `Host`
HTTP header should be resolved. Overrides global setting.
If `Paths` are defined, they take priority over DNSLink.

Default: `false` (DNSLink lookup enabled by default for every defined hostname)

Type: `bool`

`Gateway.PublicGateways: InlineDNSLink`

An optional flag to explicitly configure whether subdomain gateway's redirects
(enabled by `UseSubdomains: true`) should always inline a DNSLink name (FQDN)
into a single DNS label:

```
//example.com/ipns/example.net → HTTP 301 → //example-net.ipns.example.com
```

DNSLink name inlining allows for HTTPS on public subdomain gateways with single
label wildcard TLS certs (also enabled when passing `X-Forwarded-Proto: https`),
and provides disjoint Origin per root CID when special rules like
https://publicsuffix.org, or a custom localhost logic in browsers like Brave
has to be applied.

Default: `false`

Type: `flag`

Implicit defaults of `Gateway.PublicGateways`

Default entries for `localhost` hostname and loopback IPs are always present.
If additional config is provided for those hostnames, it will be merged on top of implicit values:

```
{
  "Gateway": {
    "PublicGateways": {
      "localhost": {
        "Paths": ["/ipfs", "/ipns"],
        "UseSubdomains": true
      }
    }
  }
}
```

It is also possible to remove a default by setting it to `null`.

For example, to disable subdomain gateway on `localhost`
and make that hostname act the same as `127.0.0.1`:

```
$ ipfs config --json Gateway.PublicGateways '{"localhost": null }'
```

`Gateway` recipes

Below is a list of the most common public gateway setups.

- Public subdomain gateway at `http://{cid}.ipfs.dweb.link` (each content root gets its own Origin)

  ```
  $ ipfs config --json Gateway.PublicGateways '{
      "dweb.link": {
        "UseSubdomains": true,
        "Paths": ["/ipfs", "/ipns"]
      }
    }'
  ```

  - **Backward-compatible:** this feature enables automatic redirects from content paths to subdomains:

    `http://dweb.link/ipfs/{cid}` → `http://{cid}.ipfs.dweb.link`

  - **X-Forwarded-Proto:** if you run Kubo behind a reverse proxy that provides TLS, make it add a `X-Forwarded-Proto: https` HTTP header to ensure users are redirected to `https://`, not `http://`. It will also ensure DNSLink names are inlined to fit in a single DNS label, so they work fine with a wildcart TLS cert (details). The NGINX directive is `proxy_set_header X-Forwarded-Proto "https";`.:

`http://dweb.link/ipfs/{cid}` → `https://{cid}.ipfs.dweb.link`

`http://dweb.link/ipns/your-dnslink.site.example.com` → `https://your--dnslink-site-example-com.ipfs.dweb.link`

  - **X-Forwarded-Host:** we also support `X-Forwarded-Host: example.com` if you want to override subdomain gateway host from the original request:

    `http://dweb.link/ipfs/{cid}` → `http://{cid}.ipfs.example.com`

- Public path gateway at `http://ipfs.io/ipfs/{cid}` (no Origin separation)

```
$ ipfs config --json Gateway.PublicGateways '{
    "ipfs.io": {
      "UseSubdomains": false,
      "Paths": ["/ipfs", "/ipns", "/api"]
    }
  }'
```

- Public DNSLink gateway resolving every hostname passed in `Host` header.

```
$ ipfs config --json Gateway.NoDNSLink false
```

  - Note that `NoDNSLink: false` is the default (it works out of the box unless set to `true` manually)

- Hardened, site-specific DNSLink gateway.

  Disable fetching of remote data (`NoFetch: true`) and resolving DNSLink at unknown hostnames (`NoDNSLink: true`).
  Then, enable DNSLink gateway only for the specific hostname (for which data
  is already present on the node), without exposing any content-addressing `Paths`:

```
$ ipfs config --json Gateway.NoFetch true
$ ipfs config --json Gateway.NoDNSLink true
$ ipfs config --json Gateway.PublicGateways '{
    "en.wikipedia-on-ipfs.org": {
      "NoDNSLink": false,
      "Paths": []
    }
  }'
```

`Identity`

`Identity.PeerID`

The unique PKI identity label for this configs peer. Set on init and never read,
it's merely here for convenience. Ipfs will always generate the peerID from its
keypair at runtime.

Type: `string` (peer ID)

`Identity.PrivKey`

The base64 encoded protobuf describing (and containing) the node's private key.

Type: `string` (base64 encoded)

`Internal`

This section includes internal knobs for various subsystems to allow advanced users with big or private infrastructures to fine-tune some behaviors without the need to recompile Kubo.

Be aware that making informed change here requires in-depth knowledge and most users should leave these untouched. All knobs listed here are subject to breaking changes between versions.

`Internal.Bitswap`

`Internal.Bitswap` contains knobs for tuning bitswap resource utilization.
The knobs (below) document how their value should related to each other.
Whether their values should be raised or lowered should be determined
based on the metrics `ipfs_bitswap_active_tasks`, `ipfs_bitswap_pending_tasks`,
`ipfs_bitswap_pending_block_tasks` and `ipfs_bitswap_active_block_tasks`
reported by bitswap.

These metrics can be accessed as the Prometheus endpoint at
`{Addresses.API}/debug/metrics/prometheus` (default:
`http://127.0.0.1:5001/debug/metrics/prometheus`)

The value of `ipfs_bitswap_active_tasks` is capped by `EngineTaskWorkerCount`.

The value of `ipfs_bitswap_pending_tasks` is generally capped by the knobs below,
however its exact maximum value is hard to predict as it depends on task sizes
as well as number of requesting peers. However, as a rule of thumb,
during healthy operation this value should oscillate around a "typical" low value
(without hitting a plateau continuously).

If `ipfs_bitswap_pending_tasks` is growing while `ipfs_bitswap_active_tasks` is at its maximum then the node has reached its resource limits and new requests are unable to be processed as quickly as they are coming in.
Raising resource limits (using the knobs below) could help, assuming the hardware can support the new limits.

The value of `ipfs_bitswap_active_block_tasks` is capped by `EngineBlockstoreWorkerCount`.

The value of `ipfs_bitswap_pending_block_tasks` is indirectly capped by `ipfs_bitswap_active_tasks`, but can be hard to
predict as it depends on the number of blocks involved in a peer task which can vary.

If the value of `ipfs_bitswap_pending_block_tasks` is observed to grow,
while `ipfs_bitswap_active_block_tasks` is at its maximum, there is indication that the number of available block tasks is creating a bottleneck (either due to high-latency block operations,
or due to high number of block operations per bitswap peer task).
In such cases, try increasing the `EngineBlockstoreWorkerCount`.
If this adjustment still does not increase the throughput of the node, there might
be hardware limitations like I/O or CPU.

`Internal.Bitswap.TaskWorkerCount`

Number of threads (goroutines) sending outgoing messages.
Throttles the number of concurrent send operations.

Type: `optionalInteger` (thread count, `null` means default which is 8)

`Internal.Bitswap.EngineBlockstoreWorkerCount`

Number of threads for blockstore operations.
Used to throttle the number of concurrent requests to the block store.
The optimal value can be informed by the metrics `ipfs_bitswap_pending_block_tasks` and `ipfs_bitswap_active_block_tasks`.
This would be a number that depends on your hardware (I/O and CPU).

Type: `optionalInteger` (thread count, `null` means default which is 128)

`Internal.Bitswap.EngineTaskWorkerCount`

Number of worker threads used for preparing and packaging responses before they are sent out.
This number should generally be equal to `TaskWorkerCount`.

Type: `optionalInteger` (thread count, `null` means default which is 8)

`Internal.Bitswap.MaxOutstandingBytesPerPeer`

Maximum number of bytes (across all tasks) pending to be processed and sent to any individual peer.
This number controls fairness and can vary from 250Kb (very fair) to 10Mb (less fair, with more work dedicated to peers who ask for more). Values below 250Kb could cause thrashing.
Values above 10Mb open the potential for aggressively-wanting peers to consume all resources and deteriorate the quality provided to less aggressively-wanting peers.

Type: `optionalInteger` (byte count, `null` means default which is 1MB)

`Internal.Bitswap.ProviderSearchDelay`

This parameter determines how long to wait before looking for providers outside of bitswap.
Other routing systems like the DHT are able to provide results in less than a second, so lowering
this number will allow faster peers lookups in some cases.

Type: `optionalDuration` (`null` means default which is 1s)

`Internal.UnixFSShardingSizeThreshold`

The sharding threshold used internally to decide whether a UnixFS directory should be sharded or not.
This value is not strictly related to the size of the UnixFS directory block and any increases in
the threshold should come with being careful that block sizes stay under 2MiB in order for them to be
reliably transferable through the networking stack (IPFS peers on the public swarm tend to ignore requests for
blocks bigger than 2MiB).

Decreasing this value to 1B is functionally equivalent to the previous experimental sharding option to
shard all directories.

Type: `optionalBytes` (`null` means default which is 256KiB)

`Ipns`

`Ipns.RepublishPeriod`

A time duration specifying how frequently to republish ipns records to ensure
they stay fresh on the network.

Default: 4 hours.

Type: `interval` or an empty string for the default.

`Ipns.RecordLifetime`

A time duration specifying the value to set on ipns records for their validity
lifetime.

Default: 24 hours.

Type: `interval` or an empty string for the default.

`Ipns.ResolveCacheSize`

The number of entries to store in an LRU cache of resolved ipns entries. Entries
will be kept cached until their lifetime is expired.

Default: `128`

Type: `integer` (non-negative, 0 means the default)

`Ipns.UsePubsub`

Enables IPFS over pubsub experiment for publishing IPNS records in real time.

EXPERIMENTAL: read about current limitations at experimental-features.md#ipns-pubsub.

Default: `disabled`

Type: `flag`

`Migration`

Migration configures how migrations are downloaded and if the downloads are added to IPFS locally.

`Migration.DownloadSources`

Sources in order of preference, where "IPFS" means use IPFS and "HTTPS" means use default gateways. Any other values are interpreted as hostnames for custom gateways. An empty list means "use default sources".

Default: `["HTTPS", "IPFS"]`

`Migration.Keep`

Specifies whether or not to keep the migration after downloading it. Options are "discard", "cache", "pin". Empty string for default.

Default: `cache`

`Mounts`

EXPERIMENTAL: read about current limitations at fuse.md.

FUSE mount point configuration options.

`Mounts.IPFS`

Mountpoint for `/ipfs/`.

Default: `/ipfs`

Type: `string` (filesystem path)

`Mounts.IPNS`

Mountpoint for `/ipns/`.

Default: `/ipns`

Type: `string` (filesystem path)

`Mounts.FuseAllowOther`

Sets the 'FUSE allow other'-option on the mount point.

`Pinning`

Pinning configures the options available for pinning content
(i.e. keeping content longer-term instead of as temporarily cached storage).

`Pinning.RemoteServices`

`RemoteServices` maps a name for a remote pinning service to its configuration.

A remote pinning service is a remote service that exposes an API for managing
that service's interest in long-term data storage.

The exposed API conforms to the specification defined at
https://ipfs.github.io/pinning-services-api-spec/

`Pinning.RemoteServices: API`

Contains information relevant to utilizing the remote pinning service

Example:

```
{
  "Pinning": {
    "RemoteServices": {
      "myPinningService": {
        "API" : {
          "Endpoint" : "https://pinningservice.tld:1234/my/api/path",
          "Key" : "someOpaqueKey"
				}
      }
    }
  }
}
```

`Pinning.RemoteServices: API.Endpoint`

The HTTP(S) endpoint through which to access the pinning service

Example: "https://pinningservice.tld:1234/my/api/path"

Type: `string`

`Pinning.RemoteServices: API.Key`

The key through which access to the pinning service is granted

Type: `string`

`Pinning.RemoteServices: Policies`

Contains additional opt-in policies for the remote pinning service.

`Pinning.RemoteServices: Policies.MFS`

When this policy is enabled, it follows changes to MFS
and updates the pin for MFS root on the configured remote service.

A pin request to the remote service is sent only when MFS root CID has changed
and enough time has passed since the previous request (determined by `RepinInterval`).

One can observe MFS pinning details by enabling debug via `ipfs log level remotepinning/mfs debug`
and switching back to `error` when done.

`Pinning.RemoteServices: Policies.MFS.Enabled`

Controls if this policy is active.

Default: `false`

Type: `bool`

`Pinning.RemoteServices: Policies.MFS.PinName`

Optional name to use for a remote pin that represents the MFS root CID.
When left empty, a default name will be generated.

Default: `"policy/{PeerID}/mfs"`, e.g. `"policy/12.../mfs"`

Type: `string`

`Pinning.RemoteServices: Policies.MFS.RepinInterval`

Defines how often (at most) the pin request should be sent to the remote service.
If left empty, the default interval will be used. Values lower than `1m` will be ignored.

Default: `"5m"`

Type: `duration`

`Pubsub`

DEPRECATED: See #9717

Pubsub configures the `ipfs pubsub` subsystem. To use, it must be enabled by passing the `--enable-pubsub-experiment` flag to the daemon
or via the `Pubsub.Enabled` flag below.

`Pubsub.Enabled`

DEPRECATED: See #9717

Enables the pubsub system.

Default: `false`

Type: `flag`

`Pubsub.Router`

DEPRECATED: See #9717

Sets the default router used by pubsub to route messages to peers. This can be one of:

- `"floodsub"` - floodsub is a basic router that simply *floods* messages to all connected peers. This router is extremely inefficient but *very* reliable.

- `"gossipsub"` - gossipsub is a more advanced routing algorithm that will build an overlay mesh from a subset of the links in the network.

Default: `"gossipsub"`

Type: `string` (one of `"floodsub"`, `"gossipsub"`, or `""` (apply default))

`Pubsub.DisableSigning`

DEPRECATED: See #9717

Disables message signing and signature verification. Enable this option if you're operating in a completely trusted network.

It is *not* safe to disable signing even if you don't care *who* sent the message because spoofed messages can be used to silence real messages by intentionally re-using the real message's message ID.

Default: `false`

Type: `bool`

`Pubsub.SeenMessagesTTL`

DEPRECATED: See #9717

Controls the time window within which duplicate messages, identified by Message ID, will be identified and won't be emitted again.

A smaller value for this parameter means that Pubsub messages in the cache will be garbage collected sooner, which can result in a smaller cache. At the same time, if there are slower nodes in the network that forward older messages, this can cause more duplicates to be propagated through the network.

Conversely, a larger value for this parameter means that Pubsub messages in the cache will be garbage collected later, which can result in a larger cache for the same traffic pattern. However, it is less likely that duplicates will be propagated through the network.

Default: see `TimeCacheDuration` from go-libp2p-pubsub

Type: `optionalDuration`

`Pubsub.SeenMessagesStrategy`

DEPRECATED: See #9717

Determines how the time-to-live (TTL) countdown for deduplicating Pubsub messages is calculated.

The Pubsub seen messages cache is a LRU cache that keeps messages for up to a specified time duration. After this duration has elapsed, expired messages will be purged from the cache.

The `last-seen` cache is a sliding-window cache. Every time a message is seen again with the SeenMessagesTTL duration, its timestamp slides forward. This keeps frequently occurring messages cached and prevents them from being continually propagated, especially because of issues that might increase the number of duplicate messages in the network.

The `first-seen` cache will store new messages and purge them after the SeenMessagesTTL duration, even if they are seen multiple times within this duration.

Default: `last-seen` (see go-libp2p-pubsub)

Type: `optionalString`

`Peering`

Configures the peering subsystem. The peering subsystem configures Kubo to connect to, remain connected to, and reconnect to a set of nodes. Nodes should use this subsystem to create "sticky" links between frequently useful peers to improve reliability.

Use-cases:

- An IPFS gateway connected to an IPFS cluster should peer to ensure that the gateway can always fetch content from the cluster.
- A dapp may peer embedded Kubo nodes with a set of pinning services or textile cafes/hubs.
- A set of friends may peer to ensure that they can always fetch each other's content.

When a node is added to the set of peered nodes, Kubo will:

1. Protect connections to this node from the connection manager. That is, Kubo will never automatically close the connection to this node and connections to this node will not count towards the connection limit.
2. Connect to this node on startup.
3. Repeatedly try to reconnect to this node if the last connection dies or the node goes offline. This repeated re-connect logic is governed by a randomized exponential backoff delay ranging from ~5 seconds to ~10 minutes to avoid repeatedly reconnect to a node that's offline.

Peering can be asymmetric or symmetric:

- When symmetric, the connection will be protected by both nodes and will likely be very stable.
- When asymmetric, only one node (the node that configured peering) will protect the connection and attempt to re-connect to the peered node on disconnect. If the peered node is under heavy load and/or has a low connection limit, the connection may flap repeatedly. Be careful when asymmetrically peering to not overload peers.

`Peering.Peers`

The set of peers with which to peer.

```
{
  "Peering": {
    "Peers": [
      {
        "ID": "QmPeerID1",
        "Addrs": ["/ip4/18.1.1.1/tcp/4001"]
      },
      {
        "ID": "QmPeerID2",
        "Addrs": ["/ip4/18.1.1.2/tcp/4001", "/ip4/18.1.1.2/udp/4001/quic"]
      }
    ]
  }
  ...
}
```

Where `ID` is the peer ID and `Addrs` is a set of known addresses for the peer. If no addresses are specified, the DHT will be queried.

Additional fields may be added in the future.

Default: empty.

Type: `array[peering]`

`Reprovider`

`Reprovider.Interval`

Sets the time between rounds of reproviding local content to the routing system. If unset, it defaults to 12 hours. If set to the value `"0"` it will disable content reproviding.

Note: disabling content reproviding will result in other nodes on the network not being able to discover that you have the objects that you have. If you want to have this disabled and keep the network aware of what you have, you must manually announce your content periodically.

Type: `duration`

`Reprovider.Strategy`

Tells reprovider what should be announced. Valid strategies are:

- `"all"` - announce all CIDs of stored blocks
- `"pinned"` - only announce pinned CIDs recursively (both roots and child blocks)

- `"roots"` - only announce the root block of explicitly pinned CIDs

Default: `"all"`

Type: `string` (or unset for the default, which is "all")

`Routing`

Contains options for content, peer, and IPNS routing mechanisms.

`Routing.Type`

There are multiple routing options: "auto", "autoclient", "none", "dht", "dhtclient", and "custom".

- **DEFAULT:** If unset, or set to "auto", your node will use the IPFS DHT
  and parallel HTTP routers listed below for additional speed.

- If set to "autoclient", your node will behave as in "auto" but without running a DHT server.

- If set to "none", your node will use *no* routing system. You'll have to
  explicitly connect to peers that have the content you're looking for.

- If set to "dht" (or "dhtclient"/"dhtserver"), your node will ONLY use the IPFS DHT (no HTTP routers).

- If set to "custom", all default routers are disabled, and only ones defined in `Routing.Routers` will be used.

When the DHT is enabled, it can operate in two modes: client and server.

- In server mode, your node will query other peers for DHT records, and will
  respond to requests from other peers (both requests to store records and
  requests to retrieve records).

- In client mode, your node will query the DHT as a client but will not respond
  to requests from other peers. This mode is less resource-intensive than server
  mode.

When `Routing.Type` is set to `auto` or `dht`, your node will start as a DHT client, and
switch to a DHT server when and if it determines that it's reachable from the
public internet (e.g., it's not behind a firewall).

To force a specific DHT-only mode, client or server, set `Routing.Type` to
`dhtclient` or `dhtserver` respectively. Please do not set this to `dhtserver`
unless you're sure your node is reachable from the public network.

When `Routing.Type` is set to `auto` or `autoclient` your node will accelerate some types of routing
by leveraging HTTP endpoints compatible with IPIP-337
in addition to the IPFS DHT.
By default, an instance of IPNI
at https://cid.contact is used.
Alternative routing rules can be configured in `Routing.Routers` after setting `Routing.Type` to `custom`.

Default: `auto` (DHT + IPNI)

Type: `optionalString` (`null`/missing means the default)

`Routing.Routers`

EXPERIMENTAL: `Routing.Routers` configuration may change in future release

Map of additional Routers.

Allows for extending the default routing (DHT) with alternative Router implementations.

The map key is a name of a Router, and the value is its configuration.

Default: `{}`

Type: `object[string->object]`

`Routing.Routers: Type`

EXPERIMENTAL: `Routing.Routers` configuration may change in future release

It specifies the routing type that will be created.

Currently supported types:

- `http` simple delegated routing based on HTTP protocol from IPIP-337
- `dht` provides decentralized routing based on libp2p's kad-dht
- `parallel` and `sequential`: Helpers that can be used to run several routers sequentially or in parallel.

Type: `string`

`Routing.Routers: Parameters`

EXPERIMENTAL: `Routing.Routers` configuration may change in future release

Parameters needed to create the specified router. Supported params per router type:

HTTP:

- `Endpoint` (mandatory): URL that will be used to connect to a specified router.
- `MaxProvideBatchSize`: This number determines the maximum amount of CIDs sent per batch. Servers might not accept more than 100 elements per batch. 100 elements by default.
- `MaxProvideConcurrency`: It determines the number of threads used when providing content. GOMAXPROCS by default.

DHT:

- `"Mode"`: Mode used by the DHT. Possible values: "server", "client", "auto"
- `"AcceleratedDHTClient"`: Set to `true` if you want to use the experimentalDHT.
- `"PublicIPNetwork"`: Set to `true` to create a `WAN` DHT. Set to `false` to create a `LAN` DHT.

Parallel:

- `Routers`: A list of routers that will be executed in parallel:
  - `Name:string`: Name of the router. It should be one of the previously added to `Routers` list.
  - `Timeout:duration`: Local timeout. It accepts strings compatible with Go `time.ParseDuration(string)` (`10s`, `1m`, `2h`). Time will start counting when this specific router is called, and it will stop when the router returns, or we reach the specified timeout.
  - `ExecuteAfter:duration`: Providing this param will delay the execution of that router at the specified time. It accepts strings compatible with Go `time.ParseDuration(string)` (`10s`, `1m`, `2h`).
  - `IgnoreErrors:bool`: It will specify if that router should be ignored if an error occurred.
- `Timeout:duration`: Global timeout. It accepts strings compatible with Go `time.ParseDuration(string)` (`10s`, `1m`, `2h`).

Sequential:

- `Routers`: A list of routers that will be executed in order:
  - `Name:string`: Name of the router. It should be one of the previously added to `Routers` list.
  - `Timeout:duration`: Local timeout. It accepts strings compatible with Go `time.ParseDuration(string)`. Time will start counting when this specific router is called, and it will stop when the router returns, or we reach the specified timeout.
  - `IgnoreErrors:bool`: It will specify if that router should be ignored if an error occurred.
- `Timeout:duration`: Global timeout. It accepts strings compatible with Go `time.ParseDuration(string)`.

Default: `{}` (use the safe implicit defaults)

Type: `object[string->string]`

`Routing: Methods`

`Methods:map` will define which routers will be executed per method. The key will be the name of the method: `"provide"`, `"find-providers"`, `"find-peers"`, `"put-ipns"`, `"get-ipns"`. All methods must be added to the list.

The value will contain:

- `RouterName:string`: Name of the router. It should be one of the previously added to `Routing.Routers` list.

Type: `object[string->object]`

Examples:

Complete example using 2 Routers, DHT (LAN/WAN) and parallel.

```
$ ipfs config Routing.Type --json '"custom"'

$ ipfs config Routing.Routers.WanDHT --json '{
  "Type": "dht",
  "Parameters": {
    "Mode": "auto",
    "PublicIPNetwork": true,
    "AcceleratedDHTClient": false
  }
}'

$ ipfs config Routing.Routers.LanDHT --json '{
  "Type": "dht",
  "Parameters": {
    "Mode": "auto",
    "PublicIPNetwork": false,
    "AcceleratedDHTClient": false
  }
}'

$ ipfs config Routing.Routers.ParallelHelper --json '{
  "Type": "parallel",
  "Parameters": {
    "Routers": [
        {
        "RouterName" : "LanDHT",
        "IgnoreErrors" : true,
        "Timeout": "3s"
        },
        {
        "RouterName" : "WanDHT",
        "IgnoreErrors" : false,
        "Timeout": "5m",
        "ExecuteAfter": "2s"
        }
    ]
  }
}'

ipfs config Routing.Methods --json '{
      "find-peers": {
        "RouterName": "ParallelHelper"
      },
      "find-providers": {
        "RouterName": "ParallelHelper"
      },
      "get-ipns": {
        "RouterName": "ParallelHelper"
      },
      "provide": {
        "RouterName": "ParallelHelper"
      },
      "put-ipns": {
        "RouterName": "ParallelHelper"
      }
    }'
```

Swarm

Options for configuring the swarm.

`Swarm.AddrFilters`

An array of addresses (multiaddr netmasks) to not dial. By default, IPFS nodes advertise *all* addresses, even internal ones. This makes it easier for nodes on the same network to reach each other. Unfortunately, this means that an IPFS node will try to connect to one or more private IP addresses whenever dialing another node, even if this other node is on a different network. This may trigger netscan alerts on some hosting providers or cause strain in some setups.

The `server` configuration profile fills up this list with sensible defaults, preventing dials to all non-routable IP addresses (e.g., `/ip4/192.168.0.0/ipcidr/16`, which is the multiaddress representation of `192.168.0.0/16`) but you should always check settings against your own network and/or hosting provider.

Default: `[]`

Type: `array[string]`

`Swarm.DisableBandwidthMetrics`

A boolean value that when set to true, will cause ipfs to not keep track of bandwidth metrics. Disabling bandwidth metrics can lead to a slight performance improvement, as well as a reduction in memory usage.

Default: `false`

Type: `bool`

`Swarm.DisableNatPortMap`

Disable automatic NAT port forwarding.

When not disabled (default), Kubo asks NAT devices (e.g., routers), to open up an external port and forward it to the port Kubo is running on. When this works (i.e., when your router supports NAT port forwarding), it makes the local Kubo node accessible from the public internet.

Default: `false`

Type: `bool`

`Swarm.EnableHolePunching`

Enable hole punching for NAT traversal when port forwarding is not possible.

When enabled, Kubo will coordinate with the counterparty using
a relayed connection,
to upgrade to a direct connection
through a NAT/firewall whenever possible.
This feature requires `Swarm.RelayClient.Enabled` to be set to `true`.

Default: `true`

Type: `flag`

`Swarm.EnableAutoRelay`

REMOVED

See `Swarm.RelayClient` instead.

`Swarm.RelayClient`

Configuration options for the relay client to use relay services.

Default: `{}`

Type: `object`

`Swarm.RelayClient.Enabled`

Enables "automatic relay user" mode for this node.

Your node will automatically *use* public relays from the network if it detects
that it cannot be reached from the public internet (e.g., it's behind a
firewall) and get a `/p2p-circuit` address from a public relay.

Default: `true`

Type: `flag`

`Swarm.RelayClient.StaticRelays`

Your node will use these statically configured relay servers (V1 or V2)
instead of discovering public relays V2 from the network.

Default: `[]`

Type: `array[string]`

`Swarm.RelayService`

Configuration options for the relay service that can be provided to *other* peers
on the network (Circuit Relay v2).

Default: `{}`

Type: `object`

`Swarm.RelayService.Enabled`

Enables providing `/p2p-circuit` v2 relay service to other peers on the network.

NOTE: This is the service/server part of the relay system.
Disabling this will prevent this node from running as a relay server.
Use `Swarm.RelayClient.Enabled` for turning your node into a relay user.

Default: `true`

Type: `flag`

`Swarm.RelayService.Limit`

Limits applied to every relayed connection.

Default: `{}`

Type: `object[string -> string]`

`Swarm.RelayService.ConnectionDurationLimit`

Time limit before a relayed connection is reset.

Default: `"2m"`

Type: `duration`

`Swarm.RelayService.ConnectionDataLimit`

Limit of data relayed (in each direction) before a relayed connection is reset.

Default: `131072` (128 kb)

Type: `optionalInteger`

`Swarm.RelayService.ReservationTTL`

Duration of a new or refreshed reservation.

Default: `"1h"`

Type: `duration`

`Swarm.RelayService.MaxReservations`

Maximum number of active relay slots.

Default: `128`

Type: `optionalInteger`

`Swarm.RelayService.MaxCircuits`

Maximum number of open relay connections for each peer.

Default: `16`

Type: `optionalInteger`

`Swarm.RelayService.BufferSize`

Size of the relayed connection buffers.

Default: `2048`

Type: `optionalInteger`

`Swarm.RelayService.MaxReservationsPerPeer`

Maximum number of reservations originating from the same peer.

Default: `4`

Type: `optionalInteger`

`Swarm.RelayService.MaxReservationsPerIP`

Maximum number of reservations originating from the same IP.

Default: `8`

Type: `optionalInteger`

`Swarm.RelayService.MaxReservationsPerASN`

Maximum number of reservations originating from the same ASN.

Default: `32`

Type: `optionalInteger`

`Swarm.EnableRelayHop`

REMOVED

Replaced with `Swarm.RelayService.Enabled`.

`Swarm.DisableRelay`

REMOVED

Set `Swarm.Transports.Network.Relay` to `false` instead.

`Swarm.EnableAutoNATService`

REMOVED

Please use `AutoNAT.ServiceMode`.

`Swarm.ConnMgr`

The connection manager determines which and how many connections to keep and can be configured to keep. Kubo currently supports two connection managers:

- none: never close idle connections.
- basic: the default connection manager.

By default, this section is empty and the implicit defaults defined below are used.

`Swarm.ConnMgr.Type`

Sets the type of connection manager to use, options are: `"none"` (no connection management) and `"basic"`.

Default: "basic".

Type: `optionalString` (default when unset or empty)

Basic Connection Manager

The basic connection manager uses a "high water", a "low water", and internal scoring to periodically close connections to free up resources. When a node using the basic connection manager reaches `HighWater` idle connections, it will close the least useful ones until it reaches `LowWater` idle connections.

The connection manager considers a connection idle if:

- It has not been explicitly *protected* by some subsystem. For example, Bitswap will protect connections to peers from which it is actively downloading data, the DHT will protect some peers for routing, and the peering subsystem will protect all "peered" nodes.
- It has existed for longer than the `GracePeriod`.

Example:

```json
{
  "Swarm": {
    "ConnMgr": {
      "Type": "basic",
      "LowWater": 100,
      "HighWater": 200,
      "GracePeriod": "30s"
    }
  }
}
```

`Swarm.ConnMgr.LowWater`

LowWater is the number of connections that the basic connection manager will
trim down to.

Default: `32`

Type: `optionalInteger`

`Swarm.ConnMgr.HighWater`

HighWater is the number of connections that, when exceeded, will trigger a
connection GC operation. Note: protected/recently formed connections don't count
towards this limit.

Default: `96`

Type: `optionalInteger`

`Swarm.ConnMgr.GracePeriod`

GracePeriod is a time duration that new connections are immune from being closed
by the connection manager.

Default: `"20s"`

Type: `optionalDuration`

`Swarm.ResourceMgr`

Learn more about Kubo's usage of libp2p Network Resource Manager
in the dedicated resource management docs.

`Swarm.ResourceMgr.Enabled`

Enables the libp2p Resource Manager using limits based on the defaults and/or other configuration as discussed in libp2p resource management.

Default: `true`
Type: `flag`

`Swarm.ResourceMgr.MaxMemory`

This is the max amount of memory to allow libp2p to use.
libp2p's resource manager will prevent additional resource creation while this limit is reached.
This value is also used to scale the limit on various resources at various scopes
when the default limits (discussed in libp2p resource management) are used.
For example, increasing this value will increase the default limit for incoming connections.

It is possible to inspect the runtime limits via `ipfs swarm resources --help`.

Default: `[TOTAL_SYSTEM_MEMORY]/2`
Type: `optionalBytes`

`Swarm.ResourceMgr.MaxFileDescriptors`

This is the maximum number of file descriptors to allow libp2p to use.
libp2p's resource manager will prevent additional file descriptor consumption while this limit is reached.

This param is ignored on Windows.

Default `[TOTAL_SYSTEM_FILE_DESCRIPTORS]/2`
Type: `optionalInteger`

`Swarm.ResourceMgr.Allowlist`

A list of multiaddrs that can bypass normal system limits (but are still limited by the allowlist scope).
Convenience config around go-libp2p-resource-manager#Allowlist.Add.

Default: `[]`

Type: `array[string]` (multiaddrs)

`Swarm.Transports`

Configuration section for libp2p transports. An empty configuration will apply
the defaults.

`Swarm.Transports.Network`

Configuration section for libp2p *network* transports. Transports enabled in
this section will be used for dialing. However, to receive connections on these
transports, multiaddrs for these transports must be added to `Addresses.Swarm`.

Supported transports are: QUIC, TCP, WS, Relay and WebTransport.

Each field in this section is a `flag`.

`Swarm.Transports.Network.TCP`

TCP is the most
widely used transport by Kubo nodes. It doesn't directly support encryption
and/or multiplexing, so libp2p will layer a security & multiplexing transport
over it.

Default: Enabled

Type: `flag`

Listen Addresses:

- /ip4/0.0.0.0/tcp/4001 (default)
- /ip6/::/tcp/4001 (default)

`Swarm.Transports.Network.Websocket`

Websocket is a transport usually used
to connect to non-browser-based IPFS nodes from browser-based js-ipfs nodes.

While it's enabled by default for dialing, Kubo doesn't listen on this
transport by default.

Default: Enabled

Type: `flag`

Listen Addresses:

- /ip4/0.0.0.0/tcp/4002/ws
- /ip6/::/tcp/4002/ws

`Swarm.Transports.Network.QUIC`

QUIC is a UDP-based transport with
built-in encryption and multiplexing. The primary benefits over TCP are:

1. It doesn't require a file descriptor per connection, easing the load on the OS.
2. It currently takes 2 round trips to establish a connection (our TCP transport currently takes 6).

Default: Enabled

Type: `flag`

Listen Addresses:

- /ip4/0.0.0.0/udp/4001/quic (default)
- /ip6/::/udp/4001/quic (default)

`Swarm.Transports.Network.Relay`

Libp2p Relay proxy
transport that forms connections by hopping between multiple libp2p nodes.
Allows IPFS node to connect to other peers using their `/p2p-circuit`
multiaddrs. This transport is primarily useful for bypassing firewalls and
NATs.

See also:

- Docs: Libp2p Circuit Relay
- `Swarm.RelayClient.Enabled` for getting a public
- `/p2p-circuit` address when behind a firewall.
- `Swarm.EnableHolePunching` for direct connection upgrade through relay
- `Swarm.RelayService.Enabled` for becoming a
  limited relay for other peers

Default: Enabled

Type: `flag`

Listen Addresses:

- This transport is special. Any node that enables this transport can receive inbound connections on this transport, without specifying a listen address.

`Swarm.Transports.Network.WebTransport`

A new feature of `go-libp2p`
is the WebTransport transport.

This is a spiritual descendant of WebSocket but over `HTTP/3`.
Since this runs on top of `HTTP/3` it uses `QUIC` under the hood.
We expect it to perform worst than `QUIC` because of the extra overhead,
this transport is really meant at agents that cannot do `TCP` or `QUIC` (like browsers).

WebTransport is a new transport protocol currently under development by the IETF and the W3C, and already implemented by Chrome.
Conceptually, it’s like WebSocket run over QUIC instead of TCP. Most importantly, it allows browsers to establish (secure!) connections to WebTransport servers without the need for CA-signed certificates, thereby enabling any js-libp2p node running in a browser to connect to any kubo node, with zero manual configuration involved.

The previous alternative is websocket secure, which require installing a reverse proxy and TLS certificates manually.

Default: Enabled

Type: `flag`

`Swarm.Transports.Security`

Configuration section for libp2p *security* transports. Transports enabled in this section will be used to secure unencrypted connections.

Security transports are configured with the `priority` type.

When establishing an *outbound* connection, Kubo will try each security transport in priority order (lower first), until it finds a protocol that the receiver supports. When establishing an *inbound* connection, Kubo will let the initiator choose the protocol, but will refuse to use any of the disabled transports.

Supported transports are: TLS (priority 100) and Noise (priority 300).

No default priority will ever be less than 100.

`Swarm.Transports.Security.TLS`

TLS (1.3) is the default
security transport as of Kubo 0.5.0. It's also the most scrutinized and
trusted security transport.

Default: `100`

Type: `priority`

`Swarm.Transports.Security.SECIO`

Support for SECIO has been removed. Please remove this option from your config.

`Swarm.Transports.Security.Noise`

Noise is slated to replace
TLS as the cross-platform, default libp2p protocol due to ease of
implementation. It is currently enabled by default but with low priority as it's
not yet widely supported.

Default: `300`

Type: `priority`

`Swarm.Transports.Multiplexers`

Configuration section for libp2p *multiplexer* transports. Transports enabled in
this section will be used to multiplex duplex connections.

Multiplexer transports are secured the same way security transports are, with
the `priority` type. Like with security transports, the initiator gets their
first choice.

Supported transports are: Yamux (priority 100) and Mplex (priority 200)

No default priority will ever be less than 100.

`Swarm.Transports.Multiplexers.Yamux`

Yamux is the default multiplexer used when communicating between Kubo nodes.

Default: `100`

Type: `priority`

`Swarm.Transports.Multiplexers.Mplex`

Mplex is the default multiplexer used when communicating between Kubo and all
other IPFS and libp2p implementations. Unlike Yamux:

- Mplex is a simpler protocol.
- Mplex is more efficient.
- Mplex does not have built-in keepalives.

- Mplex does not support backpressure. Unfortunately, this means that, if a single stream to a peer gets backed up for a period of time, the mplex transport will kill the stream to allow the others to proceed. On the other hand, the lack of backpressure means mplex can be significantly faster on some high-latency connections.

Default: `200`

Type: `priority`

DNS

Options for configuring DNS resolution for DNSLink and `/dns*` Multiaddrs.

`DNS.Resolvers`

Map of FQDNs to custom resolver URLs.

This allows for overriding the default DNS resolver provided by the operating system, and using different resolvers per domain or TLD (including ones from alternative, non-ICANN naming systems).

Example:

```
{
  "DNS": {
    "Resolvers": {
      "eth.": "https://dns.eth.limo/dns-query",
      "crypto.": "https://resolver.unstoppable.io/dns-query",
      "libre.": "https://ns1.iriseden.fr/dns-query",
      ".": "https://cloudflare-dns.com/dns-query"
    }
  }
}
```

Be mindful that:

- Currently only `https://` URLs for DNS over HTTPS (DoH) endpoints are supported as values.
- The default catch-all resolver is the cleartext one provided by your operating system. It can be overridden by adding a DoH entry for the DNS root indicated by `.` as illustrated above.
- Out-of-the-box support for selected decentralized TLDs relies on a centralized service which is provided on best-effort basis. The implicit DoH resolvers are:

  ```
  {
    "eth.": "https://resolver.cloudflare-eth.com/dns-query",
    "crypto.": "https://resolver.cloudflare-eth.com/dns-query"
  }
  ```

> To get all the benefits of a decentralized naming system we strongly suggest setting DoH endpoint to an empty string and running own decentralized resolver as catch-all one on localhost.

Default: `{}`

Type: `object[string -> string]`

`DNS.MaxCacheTTL`

Maximum duration for which entries are valid in the DoH cache.

This allows you to cap the Time-To-Live suggested by the DNS response (RFC2181).
If present, the upper bound is applied to DoH resolvers in `DNS.Resolvers`.

Note: this does NOT work with Go's default DNS resolver. To make this a global setting, add a `.` entry to `DNS.Resolvers` first.

Examples:

- `"5m"` DNS entries are kept for 5 minutes or less.
- `"0s"` DNS entries expire as soon as they are retrieved.

Default: Respect DNS Response TTL

Type: `optionalDuration`