

Configuring NATS Server



- [Note](#)
- [Strings and Numbers](#)
- [Variables](#)
- [Include Directive](#)
- [Configuration Reloading](#)
- [Configuration Properties](#)
 - [Connectivity](#)
 - [Clustering](#)
 - [Subject Mappings](#)
 - [Connection Timeouts](#)
 - [Limits](#)
 - [JetStream Server settings](#)
 - [JetStream Account Settings](#)
 - [Tracing open telemetry style](#)
 - [JetStream TPM encryption](#)
 - [JetStream Server Limits](#)
 - [Authentication and Authorization](#)
 - [Centralized Authentication and Authorization](#)
 - [Decentralized Authentication and Authorization](#)
 - [Runtime Configuration](#)
 - [Cluster Configuration, Monitoring and Tracing](#)

While the NATS server has many flags that allow for simple testing of features from command line. The standard way of configuring the NATS server product is through a configuration file. We use a simple configuration format that combines the best of traditional formats and newer styles such as JSON and YAML.

```
nats-server -config my-server.conf
```

The NATS configuration supports the following syntax:

- Lines can be commented with `#` and `//`
- Values can be assigned to properties with delimiters:
 - Equals sign: `foo = 2`
 - Colon: `foo: 2`
 - Whitespace: `foo 2`
- Arrays are enclosed in brackets: `["a", "b", "c"]`
- Maps are enclosed in braces: `{foo: 2}`
- Maps can be assigned with no delimiter `accounts { SYS {...}, cloud-user {...} }`
- Semicolons can be optionally used as terminators `host: 127.0.0.1; port: 4222;`

The NATS configuration file is parsed with UTF-8 encoding.

“ ”

We strongly recommend using only ASCII for names and values, limiting the use of Unicode, no ASCII text to comments.

Note

The NATS configuration in the file can also be rendered as a JSON object (with comments!), but to combine it with variables the variables still have to be unquoted.

“ ”

JSON config files should be limited machine generated configuration files

Strings and Numbers

The configuration parser is very forgiving, as you have seen:

- values can be a primitive, or a list, or a map
- strings and numbers typically do the right thing
- numbers support units such as, 1K for 1000, 1KB for 1024

String values that start with a digit *can* create issues. To force such values as strings, quote them.

BAD Config:

```
listen: 127.0.0.1:4222
authorization: {
```

```
# Bad - Number parsing error
token: 3secret
}
```

Fixed Config:

```
listen: 127.0.0.1:4222
authorization: {
  # Good
  token: "3secret"
}
```

Variables

Server configurations can specify variables. Variables allow you to reference a value from one or more sections in the configuration.

Variables syntax:

- Are block-scoped
- Are referenced with a `$` prefix. Variables in quotes blocks are ignored. For example, a usage like `foo = "$VAR1"` will result in `foo` being the literal string `"$VAR1"`.
- Variables MUST be used to be recognized as such. The config parser will distinguish `unknown field` from variable by finding a reference to the variable.
- Variable reference which are not defined will be resolved from environment variables.

Variable resolution sequence:

- Look for variable in same scope
- Look for variable in parent scopes
- Look for variable in environment variables
- If not found stop server startup with the error below

```
nats-server: variable reference for 'PORT' on line 5 can not be found
```

“ ”

If the environment variable value begins with a number you may have trouble resolving it depending on the server version you are running.

```
# Define a variable in the config
```

```
TOKEN: "secret"
```

```
# Reference the variable
```

```
authorization {  
    token: $TOKEN  
}
```

```
# Define a variable in the config
```

```
# But TOKEN is never used resulting in a config parsing error
```

```
TOKEN: "secret"
```

```
# Reference the variable
```

```
authorization {  
    token: "another secret"  
}
```

```
unknown field "TOKEN"
```

A similar configuration, but this time, the variable is resolved from the environment:

```
export TOKEN="hello"
```

```
nats-server -c /config/file
```

```
# TOKEN is defined in the environment
```

```
authorization {  
    token: $TOKEN  
}
```

Include Directive

The `include` directive allows you to split a server configuration into several files. This is useful for separating configuration into chunks that you can easily reuse between different servers.

Includes *must* use relative paths, and are relative to the main configuration (the one specified via the `-c` option):

server.conf:

```
listen: 127.0.0.1:4222
include ./auth.conf
```

“ Note that `include` is not followed by `=` or `:`, as it is a *directive*.

auth.conf:

```
authorization: {
  token: "f0oBar"
}
```

```
> nats-server -c server.conf
```

Configuration Reloading

The config file is being read by the server on startup and is not re-scanned for changes and not locked.

A server can reload most configuration changes without requiring a server restart or clients to disconnect by sending the nats-server a [signal](#):

```
nats-server --signal reload
```

As of NATS v2.10.0, a reload signal can be sent on a NATS service using a system account user, where `<server-id>` is the unique ID of the server be targeted.

```
nats --user sys --password sys request '$SYS.REQ.SERVER.<server-id>.RELOAD' ""
```

Configuration Properties

Config files have the following structure (in no specific order). All blocks and properties are optional (except host and port).

Please see sections below for links to detailed explanations of each configuration block

```
#General settings
host: 0.0.0.0
port: 4222

# Various server level options
# ...

# The following sections are maps with a set of (nested) properties

jetstream {
    # JetStream storage location, limits and encryption
    []store_dir: nats
}

tls {
    # Configuration map for tls parameters used for client connections,
    # routes and https monitoring connections.
}

gateway {
    # Configuration map for gateway. Gateways are used to connected clusters.
}

leafnodes {
    # Configuration map for leafnodes. LeafNodes are lightweight clusters.
}

mqtt {
    # Configuration map for mqtt. Allow clients to connect via mqtt protocol.
}

websocket {
    # Configuration map for websocket. Allow clients to connect via websockets.
}

accounts {
    # List of accounts and user within accounts
    # User may have an authorization and authentication section
}
```

```

authorization {
    # User may have an authorization and authentication section
    # This section is only useful when no accounts are defined
}

mappings {
    # Subject mappings for default account
    # When accounts are defined this section must be in the account map
}

resolver {
    # Pointer to external Authentication/Authorization resolver
    # There are multiple possible resolver type explained in their own chapters of this
docuemntaion
    # memory, nats-base, url ... more may be added in the future
    # This parameter can be a value `MEMORY` for simple configuration
    # or a map of properties for connecting to the resolver
}

resolver_tls {
    # TLS configuration for an URL based resolver
}

resolver_preload {
    # List of JWT tokens to be loaded at server start.
}

```

Connectivity

Property	Description	Default / Example
<code>host</code>	Host for client connections.	<code>0.0.0.0</code>
<code>port</code>	Port for client connections.	<code>4222</code>
<code>listen</code>	Listen specification <code><host>:<port></code> for client connections. Either use this or the options <code>host</code> and/or <code>port</code> .	<code>0.0.0.0:4222</code> Inherits from <code>host</code> and <code>port</code>

Property	Description	Default / Example
<code>client_advertise</code>	Alternative client listen specification <code><host>:<port></code> or just <code><host></code> to advertise to clients and other server. Advertising is only active in cluster setups with NAT. Explicitly setting the URL is useful when the server is situated behind a load balancer and/or TLS server authentication requires the correct DNS name to be presented. To completely disable <code>client_advertise</code> please set <code>no_advertise: true</code> in the cluster configuration section.	A list of all interfaces the the server is bound to. E.g. <code>127.0.0.1:4222,192.168.0.13:4222</code>
<code>tls</code>	Configuration map for tls parameters used for client connections, routes and https monitoring connections.	<code>tls {}</code> No tls active by default. Plain text TCP/IP.
<code>gateway</code>	Configuration map for gateway . Gateways are used to connected clusters into superclusters.	<code>gateway {}</code> None by default.
<code>leafnodes</code>	Configuration map for leafnodes . LeafNodes are lightweight clusters.	<code>leafnodes {}</code> None by default.
<code>mqtt</code>	Configuration map for mqtt . Allow clients to connect via mqtt protocol.	<code>mqtt {}</code> Not active by default.
<code>websocket</code>	Configuration map for websocket .	<code>websocket {}</code> Not active by default.

Clustering

Property	Description	Default
<code>cluster</code>	Configuration map for cluster . Nats Servers can form a cluster for load balancing and redundancy.	<code>cluster {}</code> Not active by default.

Subject Mappings

Note that each accounts forms its own subject namespace. Therefore the `mappings` section can appear on the server level (applying to the default account) or on the account level.

```
host: 0.0.0.0
port:4222
```

```

mappings: {
  foo: bar
}

accounts: {
  accountA: {
    mappings: {
      orders.acme.*: orders.$1
    }

    users: [
      {user: admin, password: admin},
      {user: user, password: user}
    ]
  },
}

```

Property	Description	Default
<code>mappings</code>	Configuration map for mapping subject . Allows for subjects aliasing and patterns based translation. Can be used to great effect in supercluster and leafnode configuration and when sourcing streams.	<code>mappings {}</code> (none set)

Connection Timeouts

Property	Description	Default
<code>ping_interval</code>	Duration at which pings are sent to clients, leaf nodes and routes. In the presence of client traffic, such as messages or client side pings, the server will not send pings. Therefore it is recommended to keep this value bigger than what clients use .	<code>"2m"</code>
<code>ping_max</code>	After how many unanswered pings the server will allow before closing the connection.	<code>2</code>

Property	Description	Default
<code>write_deadline</code>	Maximum number of seconds the server will block when writing. Once this threshold is exceeded the connection will be closed. See slow consumer on how to deal with this on the client.	<code>"10s"</code>

Limits

Property	Description	Default
<code>max_connections</code>	Maximum number of active client connections.	<code>64K</code>
<code>max_control_line</code>	Maximum length of a protocol line (including combined length of subject and queue group). Increasing this value may require client changes to be used. Applies to all traffic.	<code>4KB</code>
<code>max_payload</code>	Maximum number of bytes in a message payload. Reducing this size may force you to implement chunking in your clients. Applies to client and leafnode payloads. It is not recommended to use values over 8MB but <code>max_payload</code> can be set up to 64MB. The max payload must be equal to or smaller than the <code>max_pending</code> value.	<code>1MB</code>
<code>max_pending</code>	Maximum number of bytes buffered for a connection. Applies to client connections. Note that applications can also set 'PendingLimits' (number of messages and total size) for their subscriptions.	<code>64MB</code>
<code>max_subscriptions</code>	Maximum numbers of subscriptions per client and leafnode accounts connection.	<code>0</code> , unlimited

JetStream Server settings

You can enable JetStream in the server's configuration by simply adding a `jetstream {}` map. By default, the JetStream subsystem will store data in the `/tmp` directory, but you can specify the directory to use via the `store_dir`, as well as the limits for JetStream storage (a value of 0 means no limit).

Normally JetStream will be run in clustered mode and will replicate data, so the best place to store JetStream data would be locally on a fast SSD. One should specifically avoid NAS or NFS storage for JetStream.



Note that each JetStream enabled server **MUST** use its own individual storage directory. JetStream replicates data between cluster nodes (up to 5 replicas), achieving redundancy and availability through this.

JetStream does not implement standby and fault tolerance through a shared file system. If a standby server shares a storage directory with an active server, you must make sure only one is active at any time. Access conflicts are not detected. We do not recommend such a setup.

Here's an example minimal file that will store data in a local "nats" directory with some limits.

```
$ nats-server -c js.conf
```

```
jetstream {
  store_dir: nats

  # 1GB
  max_memory_store: 1073741824

  # 10GB
  max_file_store: 10737418240
}
```

Global JetStream options (server level)

Property	Description	Default	Version
enable	Enable/disable JetStream without removing this section.	true	2.2.0
store_dir	Directory to use for JetStream storage.	/tmp/nats/jetstream	2.2.0
max_memory_store	Maximum size of the 'memory' storage	75% of available memory	2.2.0
domain	Isolates the JetStream cluster to the local cluster. Recommended use with leaf nodes.	(not set)	2.2.3

Property	Description	Default	Version
<code>extension_hint</code>	<code>no_extend</code> or <code>will_extend</code> - Used in a leaf cluster that may or may not want to extend the domain of the hub it connects to. This option ONLY applies if a <code>SYSTEM</code> account is shared (connected) between leaf and hub AND the following conditions are met: * IF the leaf node is standalone (not a cluster) AND supposed to extend (by using the same domain name) the domain of the hub, <code>will_extend</code> is required to hint this. If not set, the standalone leaf node will not know it's supposed to be part of a JetStream cluster. * IF the leaf is a full cluster and is NOT supposed to extend the domain of the hub (because domain names are different) <code>no_extend</code> is recommended. This will stop the leaf cluster from waiting for the hub RAFT group to become visible. If not set in the leaf, JetStream will NOT become ready on first startup until the hub can be connected. Note that this only matters during initial start of the leaf cluster.	(not set)	2.2.3
<code>max_file_store</code>	Maximum size of the 'file' storage. For units use <code>m mb g gb t tb</code>	<code>1TB</code>	2.2.0
<code>cipher</code>	Set to enable storage-level encryption at rest . Choose either <code>chacha</code> / <code>chachapoly</code> or <code>aes</code> .	(not set)	2.3.0

Property	Description	Default	Version
<code>key</code>	The encryption key to use when encryption is enabled. A key length of at least 32 bytes is recommended. Note, this key is HMAC-256 hashed on startup which reduces the byte length to 64.	(not set)	2.3.0
<code>prev_encryption_key</code>	The previous encryption key. Used when changing storage encryption keys.	(not set)	2.10.0
<code>max_outstanding_catchup</code>	Max in-flight bytes for stream catch-up	64MB	2.9.0
<code>max_buffered_msgs</code>	Maximum number of messages JetStream will buffer in memory when falling behind with RAFT or I/O. Used to protect against OOM when there are write bursts to a queue.	10.000	2.11.0
<code>max_buffered_size</code>	Maximum number of bytes JetStream will buffer in memory when falling behind with RAFT or I/O. Used to protect against OOM when there are write bursts to a queue.	128MB	2.11.0
<code>request_queue_limit</code>	Limits the number of API commands JetStream will buffer in memory. When the limit is reached, clients will get error responses rather than a timeout. Lower the value if you want to detect clients flooding JetStream.	10.000	2.11.0

Property	Description	Default	Version
<code>sync_interval</code>	Examples: <code>10s</code> <code>1m</code> <code>always</code> - Change the default fsync/sync interval for page cache in the filestore. By default JetStream relies on stream replication in the cluster to guarantee data is available after an OS crash. If you run JetStream without replication or with a replication of just 2 you may want to shorten the fsync/sync interval. - You can force an fsync after each message with <code>always</code>, this will slow down the throughput to a few hundred msg/s. See also the documentaton about the interaction between the stream replication factor and syncing data to disk here.	2m	2.10.0
<code>strict</code>	Return errors for invalid JetStream API requests. Some older client APIs may not expect this. Set to <code>false</code> for maximum backward compatibility.	<code>true</code>	2.11.0
<code>unique_tag</code>	JetStream peers will be placed in servers with tags unique relative to the <code>unique_tag</code> prefix. E.g. nodes in a cluster (or supercluster) are tagged <code>az:1</code>, <code>az:1</code>, <code>az:2</code>, <code>az:2</code>, <code>az:3</code>, <code>az:3</code>, <code>az:3</code>. Setting <code>unique_tag=az</code> will result in a new replica 3 stream to be placed in all three availability zones. Note that the <code>unique_tag</code> MUST be the same across all nodes in a cluster AND supercluster.	(not set))	2.8.0
<code>tpm</code>	Trusted Platform Module TPM base encryption	<code>tpm {}</code> (not set)	2.11.0
<code>limits</code>	JetStream server limits	<code>limits{}</code> (not set)	2.8.0

JetStream Account Settings

A JetStream section may also appear in accounts. JetStream is disabled by default. The minimal configuration will enable JetStream.

```
accounts {  
  A {  
    jetstream {  
    }  
  }  
}
```

Property	Description	Default	Version
<code>max_memory</code>	Maximum memory for in-memory streams. Sum of all accounts must be smaller than the server limit.	no limit or server limit	2.2.0
<code>max_file</code>	Maximum memory for disk streams. Sum of all accounts must be smaller than the server limit.	no limit or server limit	2.2.0
<code>max_streams</code>	Maximum number of streams.	no limit	2.2.0
<code>max_consumers</code>	Maximum number of consumers per stream(!).	no limit	2.2.0
<code>max_ack_pending</code>	Max acks pending in explicit ack mode. Stream stops delivery when the limits have been reached. Can override the server limit.	no limit or server limit	2.8.0
<code>max_bytes_required</code>	When <code>true</code> all streams require a max_bytes limit set.	<code>false</code>	2.7.0
<code>store_max_stream_bytes</code>	Maximum size limit to which a disk stream can be set. Usually combined with <code>max_bytes_required</code>	no limit	2.8.0
<code>memory_max_stream_bytes</code>	Maximum size limit to which a memory stream can be set. Usually combined with <code>max_bytes_required</code>	no limit	2.8.0

Property	Description	Default	Version
<code>cluster_traffic</code>	<code>system</code> or <code>owner</code> Configures the account in which stream replication and RAFT traffic is sent. By default (and in all versions prior to 2.11.0) all cluster traffic was handled in the system account. When set to <code>owner</code> , such RAFT and replication traffic will be in the account where the stream was created.	<code>system</code>	2.11.0

Tracing open telemetry style

Enables tracing for messages marked with the `traceparent` header. See also: [Tracing](#)

```
accounts {
  A {
    msg_trace {
      dest: "trace-ot.accountA.samples"
      sampling: 5
    }
  }
}
```

Property	Description	Default	Version
<code>dest</code>	The tracing subject on which OT compliant messages will be sent. Tracing is disabled by default.	<code>></code>	2.2.0
<code>sampling</code>	A value between 0 and 100	no limit or server limit	2.2.0

JetStream TPM encryption

```
jetstream {
  store_dir: nats
  max_file_store: 10G
  tpm {
    keys_file: "keys"
```

```

    encryption_password: "pwd"
  }
}

```

Property	Description	Default	Version
<code>keys_file</code>	Specifies the file where encryption keys are stored. This option is required, otherwise TPM will not be active. If the file does NOT EXIST, a new key will be dynamically created and stored in the <code>pcr</code>	required	2.11.0
<code>encryption_password</code>	Password used for decrypting data the keys file. OR, the password used to seal the dynamically created key in the TPM store.	required	2.11.0
<code>srk_password</code>	The Storage Root Key (SRK) password is used to access the TPM's storage root key. The srk password is optional in TPM 2.0.	not set	2.11.0
<code>pcr</code>	Platform Configuration Registers (PCRs). 0-16 are reserved. Pick a value from 17 to 23.	22	2.11.0
<code>cipher</code>	<code>chacha</code> / <code>chachapoly</code> or <code>aes</code> .	<code>chachapoly</code>	2.11.0

JetStream Server Limits

```

jetstream {
  store_dir: nats
  max_file_store: 10G
  limits {
    max_ack_pending: 10000
    duplicate_window: 600s
  }
}

```

Property	Description	Default	Version
----------	-------------	---------	---------

<code>max_ack_pending</code>	Default max acks pending in explicit ack mode. Stream stops delivery when the limits have been reached.	no limit	2.8.0
<code>max_ha_assets</code>	Maximum number of RAFT assets (stream and consumers) which can be placed on this node. Will not affect stream/consumers with replicas=1	no limit	2.8.0
<code>max_request_batch</code>	Maximum fetch size for pull consumers. Use with caution. May break existing clients violating this limit.	no limit	2.8.0
<code>duplicate_window</code>	Maximum(!) de-duplication window of streams. Stream creation will fail if the value specified is larger than this.	no limit (but default for new streams is 120s)	2.8.0

Authentication and Authorization

Centralized Authentication and Authorization

A default NATS server will have no authentication or authorization enabled. This is useful for development and simple embedded use cases only. The default account is `$G`.

Once at least one user is configured in the authorization or accounts sections, the default `$G` account and no-authentication user are disabled. You can restore no authentication access by setting the `no_auth_user`.

Property	Description	Default
<code>authorization</code>	Configuration map for client authentication/authorization . List of user and their auth setting. This section is used when only the default account (<code>\$G</code>) is active.	<code>authorization {}</code> (not set)
<code>accounts</code>	Configuration map for multi tenancy via accounts . A list of accounts each with its own users and their auth settings. Each account forms its own subject and stream namespace, with no data shared unless explicit <code>import</code> and <code>export</code> is configured.	<code>accounts {}</code> (not set)

Property	Description	Default
<code>no_auth_user</code>	Username present in the authorization block or an <code>account</code> . A client connecting without any form of authentication will be associated with this user, its permissions and account.	(not set) - will deny unauthorized access by default if any other users are configured.

Decentralized Authentication and Authorization

The Configuration options here refer to [JWT](#) based authentication and authorization.

Property	Description
<code>operator</code>	The Json Web Token of the auth operator .
<code>resolver</code>	The built-in NATS <code>resolver</code> , <code>MEMORY</code> for static or <code>URL(<url>)</code> to use an external account server. (When the operator JWT contains an account URL, it will be used as default. In this case <code>resolver</code> is only needed to overwrite the default.)
<code>resolver_tls</code>	tls configuration map for tls connections to the resolver. (This is for an outgoing connection and therefore does not use <code>timeout</code> , <code>verify</code> and <code>map_and_verify</code>)
<code>resolver_preload</code>	Map to preload account public keys and their corresponding JWT. Keys consist of <code><account public nkey></code> , value is the <code><corresponding jwt></code> .

Runtime Configuration

Property	Description	Default
<code>disable_sublist_cache</code>	If <code>true</code> disable subscription caches for all accounts. This saves resources in situations where different subjects are used all the time.	<code>false</code> , cache enabled
<code>lame_duck_duration</code>	In lame duck mode the server rejects new clients and slowly closes client connections. After this duration is over, the server shuts down. This value cannot be set lower than 30 seconds. Start lame duck mode with: nats-server --signal ldm .	<code>"2m"</code>
<code>lame_duck_grace_period</code>	This is the duration the server waits, after entering lame duck mode, before starting to close client connections	<code>"10s"</code>

Property	Description	Default
<code>no_fast_producer_stall</code>	if <code>true</code> , the server will no longer stall the producer when attempting to deliver a message to a slow consumer but instead skip this consumer (by dropping the message for this consumer) and move to the next.	<code>false</code> the server will stall the fast producer

Cluster Configuration, Monitoring and Tracing

Property	Description	Default	Version
<code>server_name</code>	The server's name, shows up in logging. Defaults to the server's id. When JetStream is used, within a domain, all server names need to be unique.	Generated Server ID	
<code>server_tags</code>	A set of tags describing properties of the server. This will be exposed through <code>/varz</code> and can be used for system resource requests, such as placement of streams. It is recommended to use <code>key:value</code> style notation.	<code>[]</code>	
<code>server_metadata</code>	A map containing string keys and values describing metadata of the server. This will be exposed through <code>/varz</code> and can be used for system resource requests.	<code>{}</code>	2.12.0
<code>trace</code>	If <code>true</code> enable protocol trace log messages. Excludes the system account.	<code>false</code> , disabled	
<code>trace_verbose</code>	If <code>true</code> enable protocol trace log messages. Includes the system account.	<code>false</code> , disabled	
<code>debug</code>	If <code>true</code> enable debug log messages	<code>false</code> , disabled	
<code>logtime</code>	If set to <code>false</code> , log without timestamps	<code>true</code> , include timestamp	

Property	Description	Default	Version
<code>log_file</code>	Log file name, relative to...	No log file	
<code>log_size_limit</code>	Size in bytes after the log file rolls over to a new one	<code>0</code> , unlimited	
<code>logfile_max_num</code>	Set the number of rotated logs to retain.	<code>0</code> , unlimited	
<code>max_traced_msg_len</code>	Set a limit to the trace of the payload of a message.	<code>0</code> , unlimited	
<code>syslog</code>	Log to syslog.	<code>false</code> , disabled	
<code>remote_syslog</code>	Syslog server address.	(not set)	
<code>http_port</code>	http port for server monitoring.	(inactive)	
<code>http</code>	Listen specification <code><host>:<port></code> for server monitoring.	(inactive)	
<code>https_port</code>	https port for server monitoring. This is influenced by the <code>tls</code> property.	(inactive)	
<code>http_base_path</code>	base path for monitoring endpoints.	<code>/</code>	
<code>https</code>	Listen specification <code><host>:<port></code> for TLS server monitoring. Requires the <code>tls</code> section to be present.	(inactive)	
<code>system_account</code>	Name of the system account. Users of this account can subscribe to system events. See System Accounts for more details.	<code>\$SYS</code>	
<code>pid_file</code>	File containing PID, relative to ... This can serve as input to nats-server --signal	(non set)	
<code>port_file_dir</code>	Directory to write a file containing the servers' open ports to, relative to ...	(not set)	
<code>connect_error_reports</code>	Number of attempts at which a repeated failed route, gateway or leaf node connection is reported. Connect attempts are made once every second.	<code>3600</code> , approx every hour	

Property	Description	Default	Version
reconnect_error_reports	Number of failed attempts to reconnect a route, gateway or leaf node connection. Default is to report every attempt.	1, every failed attempt	

- #3
- cocobook-sync

2026-10-28 11:13:32 UTC
- cocobook-sync

2026-10-29 01:42:37 UTC