NATS Connection

Metadata	Value
Date	2023-10-12
Author	@Jarema
Status	Implemented
Tags	client, server, spec

Revision	Date	Author	Info
1	2023-10-12	@Jarema	Initial draft
2	2024-6-24	@aricart	Added protocol error section

Summary

This document describes how clients connect to the NATS server or NATS cluster. That includes topics like:

- connection process
- reconnect
- tls
- discoverability of other nodes in a cluster

Motivation

Ensuring a consistent way how Clients establish and maintain connection with the NATS server and provide consistent and predictable behaviour across the ecosystem.

Guide-level Explanation

Establishing the connection

TODO Add WebSocket flow.

Minimal example

1. Clients initiate a network connection to the Server.
2. Server responds with INFO json.
3. Client sends CONNECT json.
4. Clients and Server start to exchange PING/PONG messages to detect if the connection is alive.

Note If clients sets `protocol` field in Connect to equal or greater than 1, Server can send subsequent INFO on a ongoing connection. Client needs to handle them appropriately and update server lists and server info.

Auth flow

TODO

TLS

There are two flows available in the Server that enable TLS.

Standard NATS TLS (Explicit TLS)

This method is available in all NATS Server versions.

1. Clients initiate a network connection to the Server.
2. Server responds with INFO json.
3. If Server INFO contains `tls_required` set to `true`, or the client has a tls requirement set to `true`, the client performs a TLS upgrade.
4. Client sends CONNECT json.
5. Clients and Server start to exchange PING/PONG messages to detect if the connection is alive.

TLS First (Implicit TLS)

This method has been available since NATS Server 2.10.4

There are two prerequisites to use this method:

1. Server config has enabled `handshake_first` field in the `tls` block.
2. The client has set the `tls_first` option set to true.

handshake_first has those possible values:

- **`false`**: handshake first is disabled. Default value
- `true`: handshake first is enabled and enforced. Clients that do not use this flow will fail to connect.
- `duration` (i.e. 2s): a hybrid mode that will wait a given time, allowing the client to follow the `tls_first` flow. After the duration has expired, `INFO` is sent, enabling standard client TLS flow.
- `auto`: same as above, with some default value. By default it waits 50ms for TLS upgrade before sending the INFO.

The flow itself is flipped. TLS is established before the Server sends INFO:

1. Client initiate a network connection to the Server.
2. Client upgrades the connection to TLS.
3. Server sends INFO json.
4. Client sends CONNECT json.
5. Client and Server start to exchange PING/PONG messages to detect if the connection is alive.

Servers discovery

Note: Server will send back the info only

When Server sends back INFO. It may contain additional URLs to which the client can make connection attempts. The client should store those URLs and use them in the Reconnection Strategy.

A client should have an option to turn off using advertised URLs. By default, those URLs are used.

TODO: Add more in-depth explanation how topology discovery works.

Reconnection Strategies (In progress)

On-Demand reconnect

Client should have a way that allows users to force reconnection process. This can be useful for refreshing auth or rebalancing clients.

When triggered, client will drop connection to the current server and perform standard reconnection process. That means that all subscriptions and consumers should be resubscribed and their work resumed after successful reconnect where all reconnect options are respected.

For most clients, that means having a `reconnect` method on the Client/Connection handle.

Detecting disconnection

There are two methods that clients should use to detect disconnections:

1. Missing two consecutive PONGs from the Server (number of missing PONGs can be configured).
2. Handling errors from network connection.

Reconnect process

When the client detects disconnection, it starts to reconnect attempts with the following rules:

1. Immediate reconnect attempt
    - The client attempts to reconnect immediately after finding out it has been disconnected.
2. Exponential backoff with jitter
    - When the first reconnect fails, the backoff process should kick in. Default Jitter should also be included to avoid thundering herd problems.
3. If the Server returned additional URLs, the client should try reconnecting in random order to each Server on the list, unless randomization option is disabled in the client options.
4. Successful reconnect resets the timers
5. Upon reconnection, clients should resubscribe to all created subscriptions.

If there is any change in the connection state - connected/disconnected, the client should have some way of notifying the user about it. This can be a callback function or any other idiomatic mechanism in a given language for reporting asynchronous events.

Disconnect buffer Most clients have a buffer that will aggregate messages on the client side in case of disconnection. It will fill up the buffer and send pending messages as soon as connection is restored. If buffer will be filled before the connection is restored - publish attempts should return error noting that fact.

Reference-level Explanation

Client options

Although clients should provide sensible defaults for handling the connection, in many cases, it requires some tweaking. The below list defines what can be changed, what it means, and what the defaults are.

Ping interval

default: 2 minutes

As the client or server might not know that the connection is severed, NATS has Ping/Pong protocol. Client can set at what intervals it will send a PING to the server, expecting PONG. If two consecutive PONGs are missed, connection is marked as lost triggering reconnect attempt.

It's worth noting that shorter PING intervals can improve responsiveness of the client to network issues, but it also increases the load on the whole NATS system and the network itself with each added client.

Max Pings Out

default: 2

Sets number of allowed outstanding PONG responses for the client PINGs before marking client as disconnected and triggering reconnect.

Retry on failed initial connect

default: false

By default, if a client makes a connection attempt, if it fails, `connect` returns an error. In many scenarios, users might want to allow the first attempt to fail as long as clients continue the efforts and notify the progress.

When this option is enabled, the client should start the initial connection process and return the standard NATS connection/client handle while in background connection attempts are continued.

The client should not wait for the first connection to succeed or fail, as in some network scenarios, this can take much time. If the first attempt fails, a standard [Reconnect process] should be performed.

Max reconnects

**default: 3 / none

Specifies the number of consecutive reconnect attempts the client will make before giving up. This is useful for preventing `zombie services` from endlessly reaching the servers, but it can also be a footgun and surprise for users who do not expect that the client can give up entirely.

Connection timeout

default 5s

Specifies how long the client will wait for the network connection to be established. In some languages, this can hang eternally, and timeout mechanics might be necessary. In others, the network connection method might have a way to configure its timeout.

Custom reconnect delay

Default: none

If fine-grained control over reconnect attempts intervals is needed, this option allows users to specify one. Implementation should make sense in a given language. For example, it can be a callback `fn` `reconnect(attempt: int) -> Duration`.

Disconnect buffer

If given client supports storing messages during disconnect periods, this option allows to tweak the number of stored messages. It should also allow disable buffering entirely.

Tls required

default: false If set, the client enforces the TLS, whether the Server also requires it or not.

If `tls://` scheme is used in the connection string, this also enforces tls.

Ignore advertised servers

default: false When connecting to the Server, it may send back a list of other servers in the cluster of which it is aware. This can be very helpful for discoverability and removes the need for the client to pass all servers in `connect`, but it also may be unwanted if, for example, some servers URLs are unreachable for a given client.

Retain servers order

default: false By default, if many server addresses are passed in the connect string or array, the client will try to connect to them in random order. This helps healthy connection distribution, but if in a specific case list should be treated as a preference list, randomization may be turned off.

This function can be expressed "enable retaining order" or "disable randomization" depending on what is more idiomatic in given language.

Protocol Commands and Grammar

INFO

[LINK][LINK]

Send by the Server before or after establishing TLS, depending of flow used. It contains information about the Server, the nonce, and other server URLs to which the client can connect.

CONNECT

CONNECT

Send by the client in response to INFO. Contains information about client, including optional signature, client version and connection options.

Ping Pong

This is a mechanism to detect broken connections that may not be reported by the network connection in a given language.

If the Server sends `PING`, the client should answer with `PONG`. If the Client sends `PING`, the Server should answer with `PONG`.

If two (configurable) consecutive `PONGs are missed, the client should treat the connection as broken, and it should start reconnect attempts.

The default interval for PING is 2 minutes.

Error Handling (TODO)

The `-ERR` protocol message is an important signal for clients about things that are incorrect from the perspective of Permissions or Authorization.

A note about implementation - the current format of the errors is simple, but messages are not typed in a way that is simple for clients to understand what should happen - in many cases the server will disconnect th client. In other cases it is just a runtime error that an update in configuration at runtime may re-enable the client to do what was rejected previously. However, the client has no way to know whether the server will disconnect it or not.

In cases where the error is surfaced during connection it creates the nuance that it is difficult for the client to know if the error is recoverable (simply attempt to reconnect later) or not. In some cases a client connection will never resolve unless the number of maximum reconnect attempts is specified.

Permissions Violation

`Permissions Violation` means that the client tried to publish or subscribe on a subject for which it has no permissions. This type of error can happen or surface at any time, as changes to permissions intentionally or not can happen. This means that even if the subscription has been working, it is possible that it will not in the future if the permissions are altered.

The message will include `/(Publish|Subscription) to (\S+)/` this will indicate whether the error is related to a publish or subscription operation. Note that you should be careful in how you write your matchers as the message could change slightly or sport additional information (as you'll see below).

For publish permission errors, it's hard to notify the client at the point of failure unless the client is synchronous. But the standard async error notification should be sufficient. In the case of request reply, since there's a subscription handling the response, this means that you can search subscriptions related to request and reply subjects, and notify them via the response mechanism for the request depending on the type of operation that was rejected.

For subscription errors, a second level parse for `/using queue "(\S+)"/` will yield the `queue` if any that was used during the subscribe operation. This means that a client may have permissions on a subscription, but not in a specific queue or some other permutation of the subject/queue.

The server unfortunately doesn't make it easy for the client to know the actual subscription (SID) hosting the error but the logic for processing is simple: notify the first subscription that matches the subject and queue name (this assumes you track the subject and queue name in your internal subscription representation) - the server will send multiple error protocol messages (one per offense) so if multiple subscriptions, you will be able to notify all of them.

For subscriptions, errors are *terminal* for the subscription, as the server cancels the clients interest. so the client will never get any messages on it. It is very convenient for client user code to receive an error using some mechanism associated with the subscription in question as this will simplify the handling of the client code.

It is also useful to have some sort of Promise/Future/etc that will get resolved when a subscription closes (will not yield any more messages) - The Promise/Future can resolve to an error or void (not thrown) which the client can inspect for the reason if any why the subscription closed. Throwing an error is discouraged, as this would create a possibility of crashing the client. Clients can then use this information to perform their own error handling which may require taking the service offline if the subscription is vital for its operation.

Note that regardless of a localized error handling mechanism, you should also notify the async error handler as you don't know exactly where the client code is looking for errors.

Authorization Violation

`Authorization Violation` is sent whenever the credentials for a client are not accepted. This is followed by a server initiated disconnect. Clients will normally reconnect (depending on their connection options). If the client closes, this should be reported as the last error.

User Authentication Expired

`User Authentication Expired` protocol error happens whenever credentials for the client expire while the client is connected to the server. It is followed by a server disconnect. This error should be notified in the async handler. On reconnect the client is going to be rejected with `Authorization Violation` and follow its reconnect logic.

Account Expiration

`Account Authentication Expired` is sent whenever the account JWT expires and a client for the account is connected. This will result in a disconnect initiated by the server. On reconnect the client will be rejected with `Authorization Violation` until the account configuration is refreshed on the server. The client will follow its reconnect logic.

Secure Connection - TLS Required

`Secure Connection - TLS Required` is sent if the client is trying to connect on a server that requires TLS.

> [!IMPORTANT] The client should have done extensive ServerInfo investigation and determined that this would have been a failure when initiating the connection.

Maximum Number of Connections

`maximum connections exceeded` server limit on number of connections reached. Server will send to the client the `-ERR maximum connections exceeded`, client possibly go in reconnect loop.

The server can also send `Connection throttling is active. Please try again later.` when too many TLS connections are in progress. This should be treated as `maximum connections exceeded` or reworked on the server to send this error instead. Note that this can happen if when the tls server option `connection_rate_limit` is set.

Max Payload Violation

`Maximum Payload Violation` is sent to the client if it attempts to publish more data than it is allowed by `max_payload`. The server will disconnect the client after sending the protocol error. Note that clients should test payload sizes and fail publishes that exceed the server configuration, as this allow the error to be localized when possible to the user code that caused the error.

Maximum Subscriptions Exceeded

`maximum subscriptions exceeded` is sent to the client if attempts to create more subscriptions than it the account is allowed. The error is not terminal to the connection.

User Authentication Revoked

`User Authentication Revoked` this is reported when an account is updated and the user is revoked in the account. On connects where the user is already revoked, it is just an `Authorization Error`. On actual experimentation, the client never saw `User Authentication Revoked`, and instead was just disconnected. Reconnect was greeted with a `Authorization Error`.

Invalid Client Protocol

`invalid client protocol` sent to the client if the protocol version from the client doesn't match. Client is disconnected when this error is sent.

> [!NOTE] Currently, this is not a concern since presumably, a server will be able to deal with protocol version 1 when protocol upgrades.

No Responders Requires Headers

`no responders requires headers support` sent if the client requests no responder, but rejects headers. Client is disconnected when this error is sent. Current clients hardcode `headers: true`, so this error shouldn't be seen by clients.

> [!IMPORTANT] `headers` connect option shouldn't be exposed by the clients - this is a holdover from when clients opted in to `headers`.

Failed Account Registration

`Failed Account Registration` an internal error while registering an account. (Looking for reproducible test).

Invalid Publish Subject

`Invalid Publish Subject` (this requires the server in pedantic mode). Client is not disconnected when this error is sent. Note that for subscribe operations, depending on the separator (space) you may inadvertently specify a queue. In such cases there will be no error, your subscription will simply be part of a queue. If multiple spaces or some other variant, the server will treat it as a protocol error.

Unknown Protocol Operation

`Unknown Protocol Operation` this error is sent if the server doesn't understand a command. This is followed by a disconnect.

Other Errors (not necessarily seen by the client)

- `maximum account active connections exceeded` not notified to the client, the client connecting will be disconnected (seen as a connection refused.)

Security Considerations

Discuss any additional security considerations pertaining to the TLS implementation and connection handling.

Future Possibilities

Smart Reconnection could be a potential big improvement.