

---

## NATS - Ruby Client

A Ruby client for the NATS messaging system.

License [Apache2](#) build [passing](#) gem version [0.11.0](#) [yard](#) [docs](#)

### Getting Started

```
1 gem install nats
2
3 nats-sub foo &
4 nats-pub foo 'Hello World!'
```

Starting from v0.11.0 release, you can also optionally install NKEYS in order to use the new NATS v2.0 auth features:

```
1 gem install nkeys
```

If you're looking for a non-EventMachine alternative, check out the [nats-pure](#) gem.

### Basic Usage

```
1 require "nats/client"
2
3 NATS.start do
4
5   # Simple Subscriber
6   NATS.subscribe('foo') { |msg| puts "Msg received : '#{msg}'" }
7
8   # Simple Publisher
9   NATS.publish('foo.bar.baz', 'Hello World!')
10
11  # Unsubscribing
12  sid = NATS.subscribe('bar') { |msg| puts "Msg received : '#{msg}'" }
13  NATS.unsubscribe(sid)
14
15  # Requests
16  NATS.request('help') { |response| puts "Got a response: '#{response}'" }
17
18  # Replies
19  NATS.subscribe('help') { |msg, reply| NATS.publish(reply, "I'll help!") }
20
21  # Stop using NATS.stop, exits EM loop if NATS.start started the loop
22  NATS.stop
```

---

```
23
24 end
```

## Wildcard Subscriptions

```
1 # "*" matches any token, at any level of the subject.
2 NATS.subscribe('foo.*.baz') { |msg, reply, sub| puts "Msg received on
  [{sub}] : '#{msg}'" }
3 NATS.subscribe('foo.bar.*') { |msg, reply, sub| puts "Msg received on
  [{sub}] : '#{msg}'" }
4 NATS.subscribe('*.bar.*') { |msg, reply, sub| puts "Msg received on
  [{sub}] : '#{msg}'" }
5
6 # ">" matches any length of the tail of a subject and can only be the
  last token
7 # E.g. 'foo.>' will match 'foo.bar', 'foo.bar.baz', 'foo.foo.bar.bax
  .22'
8 NATS.subscribe('foo.>') { |msg, reply, sub| puts "Msg received on [{
  sub}] : '#{msg}'" }
```

## Queues Groups

```
1 # All subscriptions with the same queue name will form a queue group
2 # Each message will be delivered to only one subscriber per queue group
  , queuing semantics
3 # You can have as many queue groups as you wish
4 # Normal subscribers will continue to work as expected.
5 NATS.subscribe(subject, :queue => 'job.workers') { |msg| puts "Received
  '#{msg}'" }
```

## Clustered Usage

```
1 NATS.start(:servers => ['nats://127.0.0.1:4222', 'nats://127.0.0.1:4223
  ']) do |nc|
2   puts "NATS is connected to #{nc.connected_server}"
3
4   nc.on_reconnect do
5     puts "Reconnected to server at #{nc.connected_server}"
6   end
7
8   nc.on_disconnect do |reason|
9     puts "Disconnected: #{reason}"
10  end
11
```

---

```
12 nc.on_close do
13   puts "Connection to NATS closed"
14 end
15 end
16
17 opts = {
18   :dont_randomize_servers => true,
19   :reconnect_time_wait => 0.5,
20   :max_reconnect_attempts => 10,
21   :servers => ['nats://127.0.0.1:4222', 'nats://127.0.0.1:4223', 'nats
      ://127.0.0.1:4224']
22 }
23
24 NATS.connect(opts) do |c|
25   puts "NATS is connected!"
26 end
```

## Auto discovery

The client also auto discovers new nodes announced by the server as they attach to the cluster. Re-connection logic parameters such as time to back-off on failure and max attempts apply the same to both discovered nodes and those defined explicitly on connect:

```
1 opts = {
2   :dont_randomize_servers => true,
3   :reconnect_time_wait => 0.5,
4   :max_reconnect_attempts => 10,
5   :servers => ['nats://127.0.0.1:4222', 'nats://127.0.0.1:4223'],
6   :user => 'secret',
7   :pass => 'deadbeef'
8 }
9
10 NATS.connect(opts) do |c|
11   # Confirm number of available servers in cluster.
12   puts "Connected to NATS! Servers in pool: #{c.server_pool.count}"
13 end
```

## Advanced Usage

```
1 # Publish with closure, callback fires when server has processed the
  message
2 NATS.publish('foo', 'You done?') { puts 'msg processed!' }
3
4 # Timeouts for subscriptions
5 sid = NATS.subscribe('foo') { received += 1 }
6 NATS.timeout(sid, TIMEOUT_IN_SECS) { timeout_recvd = true }
```

---

```

7
8 # Timeout unless a certain number of messages have been received
9 NATS.timeout(sid, TIMEOUT_IN_SECS, :expected => 2) { timeout_recvd =
    true }
10
11 # Auto-unsubscribe after MAX_WANTED messages received
12 NATS.unsubscribe(sid, MAX_WANTED)
13
14 # Multiple connections
15 NATS.subscribe('test') do |msg|
16     puts "received msg"
17
18     # Gracefully disconnect from NATS after handling
19     # messages that have already been delivered by server.
20     NATS.drain
21 end
22
23 # Form second connection to send message on
24 NATS.connect { NATS.publish('test', 'Hello World!') }

```

See examples and benchmarks for more information..

## TLS

Advanced customizations options for setting up a secure connection can be done by including them on connect:

```

1 options = {
2   :servers => [
3     'nats://secret:deadbeef@127.0.0.1:4443',
4     'nats://secret:deadbeef@127.0.0.1:4444'
5   ],
6   :max_reconnect_attempts => 10,
7   :reconnect_time_wait => 2,
8   :tls => {
9     :private_key_file => './spec/configs/certs/key.pem',
10    :cert_chain_file => './spec/configs/certs/server.pem'
11    # Can enable verify_peer functionality optionally by passing
12    # the location of a ca_file.
13    # :verify_peer => true,
14    # :ca_file => './spec/configs/certs/ca.pem'
15  }
16 }
17
18 # Set default callbacks
19 NATS.on_error do |e|
20     puts "Error: #{e}"
21 end
22

```

---

```
23 NATS.on_disconnect do |reason|
24   puts "Disconnected: #{reason}"
25 end
26
27 NATS.on_reconnect do |nats|
28   puts "Reconnected to NATS server at #{nats.connected_server}"
29 end
30
31 NATS.on_close do
32   puts "Connection to NATS closed"
33   EM.stop
34 end
35
36 NATS.start(options) do |nats|
37   puts "Connected to NATS at #{nats.connected_server}"
38
39   nats.subscribe("hello") do |msg|
40     puts "Received: #{msg}"
41   end
42
43   nats.flush do
44     nats.publish("hello", "world")
45   end
46 end
```

## Fibers

Requests without a callback can be made to work synchronously and return the result when running in a Fiber. For these type of requests, it is possible to set a timeout of how long to wait for a single or multiple responses.

```
1 NATS.start {
2
3   NATS.subscribe('help') do |msg, reply|
4     puts "[Received]: <<- #{msg}"
5     NATS.publish(reply, "I'll help! - #{msg}")
6   end
7
8   NATS.subscribe('slow') do |msg, reply|
9     puts "[Received]: <<- #{msg}"
10    EM.add_timer(1) { NATS.publish(reply, "I'll help! - #{msg}") }
11  end
12
13  10.times do |n|
14    NATS.subscribe('hi') do |msg, reply|
15      NATS.publish(reply, "Hello World! - id:#{n}")
16    end
17  end
```

```

18
19 Fiber.new do
20   # Requests work synchronously within the same Fiber
21   # returning the message when done.
22   response = NATS.request('help', 'foo')
23   puts "[Response]: ->> '#{response}'"
24
25   # Specifying a custom timeout to give up waiting for
26   # a response.
27   response = NATS.request('slow', 'bar', timeout: 2)
28   if response.nil?
29     puts "No response after 2 seconds..."
30   else
31     puts "[Response]: ->> '#{response}'"
32   end
33
34   # Can gather multiple responses with the same request
35   # which will then return a collection with the responses
36   # that were received before the timeout.
37   responses = NATS.request('hi', 'quux', max: 10, timeout: 1)
38   responses.each_with_index do |response, i|
39     puts "[Response# #{i}]: ->> '#{response}'"
40   end
41
42   # If no replies then an empty collection is returned.
43   responses = NATS.request('nowhere', '', max: 10, timeout: 2)
44   if responses.any?
45     puts "Got #{responses.count} responses"
46   else
47     puts "No response after 2 seconds..."
48   end
49
50   NATS.stop
51 end.resume
52
53 # Multiple fibers can make requests concurrently
54 # under the same Eventmachine loop.
55 Fiber.new do
56   10.times do |n|
57     response = NATS.request('help', "help.#{n}")
58     puts "[Response]: ->> '#{response}'"
59   end
60 end.resume
61 }

```

## New Authentication (Nkeys and User Credentials)

This requires server with version  $\geq 2.0.0$

---

NATS servers have a new security and authentication mechanism to authenticate with user credentials and NKEYS. A single file containing the JWT and NKEYS to authenticate against a NATS v2 server can be set with the `user_credentials` option:

```
1 require 'nats/client'
2
3 NATS.start("tls://connect.ngs.global", user_credentials: "/path/to/
  creds") do |nc|
4   nc.subscribe("hello") do |msg|
5     puts "[Received] #{msg}"
6   end
7   nc.publish('hello', 'world')
8 end
```

This will create two callback handlers to present the user JWT and sign the nonce challenge from the server. The core client library never has direct access to your private key and simply performs the callback for signing the server challenge. The library will load and wipe and clear the objects it uses for each connect or reconnect.

Bare NKEYS are also supported. The nkey seed should be in a read only file, e.g. `seed.txt`.

```
1 > cat seed.txt
2 # This is my seed nkey!
3 SUAGMJH5XLGZKQWAWKRZJIGMOU4HPFUYLXJMX005NLFE0200QJ5LPRDPM
```

Then in the client specify the path to the seed using the `nkeys_seed` option:

```
1 require 'nats/client'
2
3 NATS.start("tls://connect.ngs.global", nkeys_seed: "path/to/seed.txt")
  do |nc|
4   nc.subscribe("hello") do |msg|
5     puts "[Received] #{msg}"
6   end
7   nc.publish('hello', 'world')
8 end
```

## License

Unless otherwise noted, the NATS source files are distributed under the Apache Version 2.0 license found in the LICENSE file.