JetStream Publish Retries on No Responders

Metadata	Value
Date	2022-03-18
Author	wallyqs
Status	Partially Implemented
Tags	jetstream, client

Motivation

When the NATS Server is running with JetStream on cluster mode, there can be occasional blips in leadership which can result in a number of `no responders available` errors during the election. In order to try to mitigate these failures, retries can be added into JetStream enabled clients to attempt to publish the message to JetStream once it is ready again.

Implementation

A `no responders available` error uses the 503 status header to signal a client that there was no one available to serve the published request. A synchronous `Publish` request when using the JetStream context internally uses a `Request` to produce a message and if the JetStream service was not ready at the moment of publishing, the server will send to the requestor a 503 status message right away.

To improve robustness of producing messages to JetStream, a client can back off for a a bit and then try to send the message again later. By default, the Go client waits for `250ms` and will retry 2 times sending the message (so that in total it would have attempted to send the message 3 times).

Below can be found an example implementation using the `Request` API from the Go client:

```
// Stream that persists messages sent to 'foo'
js.AddStream(&nats.StreamConfig{Name: "foo"})

var (
    retryWait   = 250 * time.Millisecond
    maxAttempts = 2
    i           = 0
)

// Loop to publish a message every 100ms
for range time.NewTicker(100 * time.Millisecond).C {
    subject := "foo"
    msg := fmt.Sprintf("i:%d", i)
    _, err := nc.Request(subject, []byte(msg), 1*time.Second)
    if err != nil && err == nats.ErrNoResponders {
        for attempts := 0; attempts < maxAttempts; attempts++ {
            // Backoff before retrying
```

```
            time.Sleep(retryWait)

            // Next attempt
            _, err := nc.Request(subject, []byte(msg), 1*time.Second)
            if err != nil && err == nats.ErrNoResponders {
                // Retry again
                continue
            }
        }
    }
    i++
}
```

Errors

After exhausting the number of attempts, the result should either be a timeout error in case the deadline expired or a `nats: no response from stream` error if the error from the last attempt was still a `no responders error`.

Examples

Customizing retries with `RetryWait` and `RetryAttempts`

Two options are added to customize the retry logic from the defaults:

```
_, err := js.Publish("foo", []byte("bar"), nats.RetryWait(250*time.Millisecond),
nats.RetryAttempts(10))
if err != nil {
    log.Println("Pub Error", err)
}
```

Make Publish retry as needed until deadline

It can be possible to set the maximum deadline of the retries so that the client can retry as needed. In the example below a client will attempt to publish up to 10 seconds to wait for an ack response from the server, backing off `250ms` as needed until the service is available again:

```
// Using Go context package
ctx, cancel := context.WithTimeout(context.Background(), 10*time.Second)
defer cancel()
_, err := js.Publish("foo", []byte("bar"), nats.Context(ctx),
nats.RetryWait(250*time.Millisecond), nats.RetryAttempts(-1))
if err != nil {
    log.Println("Pub Error", err)

}

// Custom AckWait
```

```
_, err := js.Publish("foo", []byte("bar"), nats.AckWait(10*time.Second),
nats.RetryWait(250*time.Millisecond), nats.RetryAttempts(-1))
if err != nil {
    log.Println("Pub Error", err)
}
```