Pull Subscribe internals

Metadata	Value
Date	2021-07-20
Author	wallyqs
Status	Partially Implemented
Tags	jetstream, client

Motivation

One of the forms of message delivery in JetStream is a pull based consumers. In this ADR it is described what is the current state of implementing consumption of messages for this type of consumers.

Context

A pull based consumer is a type of consumer that does not have a `delivery subject`, that is the server *does not know* where to send the messages. Instead, the clients have to request for the messages to be delivered as needed from the server. For example, given a stream `bar` with the consumer with a durable name `dur` (all pull subscribers have to be durable), a pull request would look like this in terms of the protocol:

```
PUB $JS.API.CONSUMER.MSG.NEXT.bar.dur _INBOX.x7tkDPDLCOEknrfB4RH1V7.UBZe2D 0
```

Request Body

There are 3 possible fields that can be presented in the json body of the request. If there is no body presented (as in the example above, all defaults are assumed.)

batch

The number of messages that the server should send. Minimum is 1. There is not a limit on the batch size at the level of the library. The account and then server may have some limits.

no_wait

A boolean value indicating to make this pull request the no wait type. See below for details. Default is false.

expires

The number of nanoseconds, from now that this pull will expire. <= 0 or not supplied means the expiration is not applied. No wait takes precedence over expires if both are supplied.

Pull(n) Requests

A request with an empty payload is identical to a pull with batch size of 1 and results in the server sending the next (1) available message, for example:

```
SUB _INBOX.example 0
+OK
PUB $JS.API.CONSUMER.MSG.NEXT.bar.dur _INBOX.example 0
+OK
MSG bar 0 $JS.ACK.bar.dur.1.9808.9808.1626818873482533000.0 4
helo
```

Note that even though the inbox used for the request was `_INBOX.example`, when the message got delivered the subject was rewritten into `bar` which is the subject of the message that is persisted in JetStream.

A pull request for the next message, will linger until there is no more interest in the subject, a client is disconnected or the batch size is filled. Each pull request will increase the `num_awaiting` counter for a consumer of inflight pull requests [2]. At most, a consumer can only have 512 inflight pull requests, though this can be changed when creating the consumer with the `max_waiting` option [1]:

```
PUB $JS.API.CONSUMER.INFO.bar.dur _INBOX.uMfJLECClHCs0CLfWF7Rsj.ds2ZxC4o 0
MSG _INBOX.uMfJLECClHCs0CLfWF7Rsj.ds2ZxC4o 1 601
{
    "type": "io.nats.jetstream.api.v1.consumer_info_response",
    "stream_name": "bar",
    "name": "dur",
    "created": "2021-07-20T21:35:11.825973Z",
    "config": {
        "durable_name": "dur",
        "deliver_policy": "all",
        "ack_policy": "explicit",
        "ack_wait": 30000000000,
        "max_deliver": -1,
        "filter_subject": "bar",
        "replay_policy": "instant",
        "max_waiting": 512,          <-- Maximum Inflight Pull/Fetch Requests [1]
        "max_ack_pending": 20000
    },
    "delivered": {
        "consumer_seq": 11561,
        "stream_seq": 11560
    },
    "ack_floor": {
        "consumer_seq": 11561,
        "stream_seq": 11560
    },
    "num_ack_pending": 0,
    "num_redelivered": 0,
    "num_waiting": 1,                <-- Inflight Pull/Fetch Requests [2]
    "num_pending": 0,
    "cluster": {
        "leader": "NCUOUH6YICRESE73CKTXESMGA4ZN4KTELXPUIE6JCRTL6IF4UWE3B2Z4"
    }
}
```

When making a pull request it is also possible to request more than one message:

```
PUB $JS.API.CONSUMER.MSG.NEXT.bar.dur _INBOX.x7tkDPDLCOEknrfB4RH1V7.OgY4M7 32
{"batch":5,"expires":4990000000}
```

Whenever a pull request times out, the count of `num_waiting` will increase for a consumer but this will eventually reset once it reaches the max waiting inflight that was configured for the pull consumer.

No Wait Pull Requests

In order to get a response from the server right away, a client can make a pull request with the `no_wait` option enabled. For example:

```
PUB $JS.API.CONSUMER.MSG.NEXT.bar.dur _INBOX.x7tkDPDLCOEknrfB4RH1V7.OgY4M7 26
{"batch":1,"no_wait":true}
```

The result of a no wait pull request is a guaranteed instant response from the server that will be either the next message or an error, where an error could be a server error such as `503` in case JS service is not available. Most commonly, the result of a no wait pull request will be a `404` no messages error:

```
HMSG _INBOX.x7tkDPDLCOEknrfB4RH1V7.OgY4M7 2  28 28
NATS/1.0 404 No Messages
```

Design

The implementation for pull subscribe uses a combination of both no wait and lingering pull requests described previously.

In the Go client, a simple example of the API looks as follows:

```
sub, err := js.PullSubscribe("stream-name", "durable")
if err != nil {
    log.Fatal(err)
}

for {
    msgs, err := sub.Fetch(1)
    if err != nil {
        log.Println("Error:", err)
        continue
    }
```

```
        for _, msg := range msgs {
            msg.Ack()
        }
    }
```

When implementing `PullSubscribe` there are two main cases to consider: `Pull(n)` and `Pull(1)`.

Pull(n)

`Pull(n)` batch requests are implemented somewhat similarly to old style requests. When making a pull request, first a no wait request is done to try to get the messages that may already be available as needed. If there are no messages (a 404 status message error by the JetStream server), then a long pull request is made:

```
SUB _INBOX.NQ2uAIXd4GoozKOTfECtIg  1
PUB $JS.API.CONSUMER.MSG.NEXT.bar.dur _INBOX.WvaJLnIXcj8Zf5SrxlHMTS 26
{"batch":5,"no_wait":true}

# Result of no wait request if there are no messages
HMSG _INBOX.WvaJLnIXcj8Zf5SrxlHMTS 8  28 28
NATS/1.0 404 No Messages

# Next pull request is a long pull request with client side timeout
PUB $JS.API.CONSUMER.MSG.NEXT.bar.dur _INBOX.NQ2uAIXd4GoozKOTfECtIg 32
{"expires":4990000000,"batch":5}
```

As part of the request payload, the `batch` field is set to the number of expected messages and `expires` is set to cancel the request `100ms` before the client side timeout. In the example above, the timeout is `5s` so in the payload `expires` is the result of `5s - 100ms` represented in nanoseconds.

After making the first request no wait request, it is also recommended to send an auto unsubscribe protocol discounting the message already received as a result of the no wait request. In case the batch request was for 5 messages, the client would auto unsubscribe after receiving 6. Since first message was an error the `UNSUB` is (batch+1) to account for that initial error status message.

```
UNSUB 1 6
```

When successful, the result of a batch request will be at least one message being delivered to the client. In case not all messages are delivered and the client times out or goes away during the pull request, it is recommended to unsubscribe to remove the interest of the awaited messages from the server, otherwise this risks the server sending messages to an inbox from a connected client that is no longer expecting messages.

```
SUB _INBOX.WvaJLnIXcj8Zf5SrxlHMTS  1
PUB $JS.API.CONSUMER.MSG.NEXT.bar.dur _INBOX.WvaJLnIXcj8Zf5SrxlHMTS 26
{"batch":5,"no_wait":true}
HMSG _INBOX.WvaJLnIXcj8Zf5SrxlHMTS 8  28 28
```

```
NATS/1.0 404 No Messages
UNSUB 1 6
PUB $JS.API.CONSUMER.MSG.NEXT.bar.dur _INBOX.WvaJLnIXcj8Zf5SrxlHMTS 32
{"expires":4990000000,"batch":5}
MSG hello 1 $JS.ACK.bar.dur.2.29034.29041.1626845015078897000.0 4
helo
# Only 1 out of 5 messages receives, so client goes away and unsubscribes.
UNSUB 1
```

Errors and Status messages handling

While receiving messages, at any point the client may instead receive an error from the server as a status message.

```
MSG hello 1 $JS.ACK.bar.dur.2.29034.29041.1626845015078897000.0 5
hello
# (System reached too many inflight pull requests condition)
HMSG _INBOX.WvaJLnIXcj8Zf5SrxlHMRI 1  32 32
NATS/1.0 408 Request Timeout
```

Whenever there is a status message, it is recommended that the error is not returned to the user as a processable message and instead handle is as the error condition:

```
for {
    msgs, err := sub.Fetch(5)
    if err != nil {
        // Any error such as:
        //
        // - client timeout
        // - request timeout (408)
        // - bad request
        // - no responders
        // - system unavailable (no current JS quorum)
        log.Println("Error:", err)
        continue
    }
    // Msgs here are never status or error messages,
    // in case there is an error the loop just breaks.
    for _, msg := range msgs {
        msg.Ack()
    }
}
```

Pull Optimization

For the case of pulling a single message, it is possible to optimize things by making the first pull request as a no wait request instead and by preparing a new style like request/response handler using a wildcard subscription. This will result in less chatty protocol and also works better with topologies where JetStream is running as a leafnode on the edge.

```
SUB _INBOX.miOJjN58koGhobmCGCWKJz.*  2
PUB $JS.API.CONSUMER.MSG.NEXT.bar.dur _INBOX.miOJjN58koGhobmCGCWKJz.asdf 26
{"batch":1,"no_wait":true}
```

Similar to `Pull(n)`, when the first no wait request fails, after the first `Pull(1)` a longer old style request is made with a unique inbox.

Note: Each pull subscriber must have its own pull request/response handler. The default implementation of new style request cannot be used for this purpose due to how the subject gets rewritten which would cause responses to be dropped.