JetStream Direct Get

Metadata	Value
Date	2022-08-03
Author	@mh, @ivan, @derekcollison, @alberto, @tbeets, @ripienaar
Status	Implemented
Tags	jetstream, client, server, 2.11

Revision	Date	Author	Info
1	2022-08-08	@tbeets	Initial design
2	2024-03-06	@ripienaar	Adds Multi and Batch behaviors for Server 2.11

Context and motivation

In initial design of JetStream, reading a message from a stream *directly*, i.e. *not* via JS Consumer delivery, was thought to be an administrative function and API. All such reads are routed to the current stream leader (and its underlying stream store) and read calls are tracked as administrative and incur tracking overhead.

In some use cases, notably key *get* on a stream that is a KV materialized view, it is desirable to both spread message read load to multiple servers (each accessing a message store local to them) and bypass administrative API overhead.

Feature: Direct Get

The JetStream *Direct Get* feature enables all stream peers (R>1), not just the stream leader, to respond to stream read calls as a service responder *queue group*. The responder sources its local message store. With Direct Get the number of servers eligible to respond to read requests is the same as the replica count of the stream.

Extended feature: MIRROR Direct Get responders

For streams that are *Direct Get* enabled and are also an upstream source of a MIRROR stream, the mirror's peer servers will also participate in the responder queue group for Direct Get calls *to the upstream*. In this manner, message read can be spread to many additional servers.

As mirrors need not be in the same cluster as the upstream, servers that respond to Direct Get requests to the upstream can be strategically placed for client latency-reduction, e.g. different geographic locations serving distributed clients. Also, read availability can be enhanced as mirrors may be available to clients when the upstream is offline.

A note on read-after-write coherency

The existing Get API `$JS.API.STREAM.MSG.GET.<stream>` provides read-after-write coherency by routing requests to a stream's current peer leader (R>1) or single server (R=1). A client that publishes a message to

stream (with ACK) is assured that a subsequent call to the Get API will return that message as the read will go a server that defines *most current*.

In contrast, *Direct Get* does not assure read-after-write coherency as responders may be non-leader stream servers (that may not have yet applied the latest consensus writes) or MIRROR downstream servers that have not yet *consumed* the latest consensus writes from upstream.

Implementation

Stream property: Allow Direct

- Stream configuration adds a new Allow Direct boolean property: `allow_direct`
- Allow Direct is always set to `true` by the server when maximum messages per subject, `max_msgs_per_subject`, is configured to be > 0 (a max limit is specified)
- If the user passes Allow Direct explicitly in stream create or edit request, the value will be overriden based on `max_msgs_per_subject`

> Allow Direct is set automatically based on the inferred use case of the stream. Maximum messages per subject is a tell-tale of a stream that is a KV bucket.

Direct Get API

When Allow Direct is true, each of the stream's servers configures a responder and subscribes to `$JS.API.DIRECT.GET.<stream>` with fixed queue group `_sys_`.

> Note: If Allow Direct is false, there will be no responder at the Direct Get API for a stream. Clients that make requests will receive no reply message and will time out.

Request

Clients may make requests with the same payload as the Get message API populating the following server struct:

```
Seq          uint64     `json:"seq,omitempty"`
LastFor      string     `json:"last_by_subj,omitempty"`
NextFor      string     `json:"next_by_subj,omitempty"`
Batch        int        `json:"batch,omitempty"`
MaxBytes     int        `json:"max_bytes,omitempty"`
StartTime    *time.Time `json:"start_time,omitempty"`
MultiLastFor []string   `json:"multi_last,omitempty"`
UpToSeq      uint64     `json:"up_to_seq,omitempty"`
UpToTime     *time.Time `json:"up_to_time,omitempty"`
```

Example request payloads:

- `{seq: number}` - get a message by sequence
- `{last_by_subj: string}` - get the last message having the subject
- `{next_by_subj: string}` - get the first message (lowest seq) having the specified subject

- `{start_time: string}` - get the first message at or newer than the time specified in RFC 3339 format (since server 2.11)
- `{seq: number, next_by_subj: string}` - get the first message with a seq >= to the input seq that has the specified subject
- `{seq: number, batch: number, next_by_subj: string}` - gets up to batch number of messages >= than seq that has specified subject
- `{seq: number, batch: number, next_by_subj: string, max_bytes: number}` - as above but limited to a maximum size of messages received in bytes
- `{multi_last: [string]}` - get the last message for each subject in the list (subjects can include wildcards)
- `{multi_last: [string], up_to_seq: number}` - get the last message for each subject in the list up to the sequence number
- `{multi_last: [string], up_to_time: string}` - get the last message for each subject in the list up to the time specified in RFC 3339 format
- `{multi_last: [string], batch: number}` - get the last message for each subject in the list up to the batch size

Subject-Appended Direct Get API

The purpose of this form is so that environments may choose to apply subject-based interest restrictions (user permissions within an account and/or cross-account export/import grants) such that only specific subjects in stream may be read (vs any subject in the stream).

When Allow Direct is true, each of the stream's servers will also subscribe to `$JS.API.DIRECT.GET.<stream>.>` with fixed queue group `_sys_`. Requests to this API endpoint will be interpreted as a shortcut for `last_by_subj` request where `last_by_subj` is derived by the token (or series of tokens) following the stream name rather than request payload.

It is an error (408) if a client calls Subject-Appended Direct Get and includes a request payload.

Batched requests

Using the `batch` and `max_bytes` keys one can request multiple messages in a single API call.

The server will send multiple messages without any flow control to the reply subject, it will send up to `max_bytes` messages. When `max_bytes` is unset the server will use the `max_pending` configuration setting or the server default (currently 64MB)

After the batch is sent a zero length payload message will be sent with the `Nats-Num-Pending` and `Nats-Last-Sequence` headers set that clients can use to determine if further batch calls are needed. It would also have the `Status` header set to `204` with the `Description` header being `EOB`.

When requests are made against servers that do not support `batch` the first response will be received and nothing will follow. Old servers can be detected by the absence of the `Nats-Num-Pending` header in the first reply.

There are 4 viable API calls for a batch. All require a batch amount greater than 0 and a subject which may include a wildcard. They also require a start sequence -or- a start time. There is a server issue under consideration requesting that if neither start sequence nor a start time is supplied that it defaults to start

sequence of 1. For now the client can optionally provide the 2 additional calls which provide the start sequence of 1 for the user.

1. get up to batch number of messages where the message sequence is >= 1 and for the specified subject
   - API: `batch: number, subject: string`
   - Request: `{"batch":3,"seq":1,"next_by_subj":"foo.>"}`
2. get up to batch number of messages where the message sequence is >= the specified sequence and for the specified subject
   - API: `batch: number, sequence: number, subject: string`
   - Request: `{"batch":3,"seq":4,"next_by_subj":"foo.>"}`
3. get up to batch number of messages where the message timestamp is >= than start time and for the specified subject
   - API: `batch: number, start time: time, subject: string`
   - Request: `{"batch":3,"start_time":"2024-11-04T23:45:02.060192000Z","next_by_subj":"foo.>"}`
4. get up to batch number of messages where the message sequence is >= than 1, for the specified subject, and limited by max bytes
   - API: `batch: number, max_bytes: number, sequence: number, subject: string`
   - Request: `{"batch":3,"max_bytes":2002,"seq":1,"next_by_subj":"foo.>"}`
5. get up to batch number of messages where the message sequence is >= than the specified sequence, for the specified subject and limited by max bytes
   - API: `batch: number, max_bytes: number, sequence: number, subject: string`
   - Request: `{"batch":3,"max_bytes":2002,"seq":4,"next_by_subj":"foo.>"}`
6. get up to batch number of messages where the message timestamp is >= than start time, for the specified subject and limited by max bytes
   - API: `batch: number, max_bytes: number, start time: time, subject: string`
   - Request: `{"batch":3,"max_bytes":2002,"start_time":"2024-11-04T23:45:02.060192000Z","next_by_subj":"foo.>"}`

Multi-subject requests

Multiple subjects can be requested in the same manner that a Batch can be requested. In this mode we support consistent point in time reads by allowing for a group of subjects to be read as they were at a point in time - assuming the stream holds enough historical data.

To help inform proper use of this feature vs just using a consumer, any multi-subject request may only allow matching up to 1024 subjects. Any more will result in a `413` status reply.

Using requests like `{"multi_last":["$KV.USERS.1234.>"]}` all the latest values for all subjects below that wildcard will be returned.

Specific data for a user could be requested using `{"multi_last":["$KV.USERS.1234.name", "$KV.USERS.1234.address"]}`. Rather than getting all the user data, only values for for two specific keys will be returned.

To facilitate consistent multi key reads, the `up_to_seq` and `up_to_time` keys can be added which will restrict the results up to a certain point in time.

Imagine we have a new bucket and we did:

```
$ nats kv put USERS 1234.name Bob                 # message seq 1
$ nats kv put USERS 1234.surname Smith            # message seq 2
$ nats kv put USERS 1234.address 1 Main Street    # message seq 3
$ nats kv put USERS 1234.address 10 Oak Lane      # message seq 4 updates message 3
```

If we did a normal multi read using `{"multi_last":["$KV.USERS.1234.>"]}` we would get the address `10 Oak Lane` returned. However, to get a record as it was at a certain point in the past we could supply the sequence or time, adding `"up_to_seq":3` to the request will return address `1 Main Street` along with the other data. Likewise, assuming there was a noticeable gap of time changing addresses, the `up_to_time` could be used as a form of temporal querying.

A `batch` parameter can be added to restrict the result set to a certain size, otherwise the server will decide when to end the batch using the same `EOB` marker message seen in Batched Mode with the addition of the `Nats-UpTo-Sequence` header.

When the server cannot send any more data it will respond, like the above Batch, with a zero-length payload message including the `Nats-Num-Pending` and `Nats-Last-Sequence` headers enabling clients to determine if further batch calls are needed. In addition, it would also have the `Status` header set to `204` with the `Description` header being `EOB`. The `Nats-UpTo-Sequence` header will be set indicating the last message in the stream that matched criteria. This number would be used in subsequent requests as the `up_to_seq` value to ensure batches of multi-gets are done around a consistent point in time.

For the multi last API, we can make 6 distinct calls:

1. get the last messages for the specified subject(s)
    - API: `subjects: []string`
    - Request: `{"multi_last":["foo.A","foo.D"]}`
2. get the last messages for the specified subject(s), where the last message is less than or equal to the up to sequence.
    - API: `subjects: []string, up_to_sequence: number`
    - Request: `{"multi_last":["foo.A","foo.D"],"up_to_seq":23}`
3. get the last messages for the specified subject(s), where the last message is less than or equal to the up to time.
    - API: `subject: []string, up_to_time: time`
    - Request: `{"multi_last":["foo.A","foo.D"],"up_to_time":"2024-11-05T00:50:25.248431300Z"}`
4. get the last messages for the specified subject(s) specified subject, limited by batch size
    - API: `batch: number, subjects: []string`
    - Request: `{"batch":2,"multi_last":["foo.A","foo.D"]}`
5. get the last messages for the specified subject(s), where the last message is less than or equal to the up to sequence, limited by batch size.
    - API: `batch: number, subjects: []string, up_to_sequence: number`
    - Request: `{"batch":2,"multi_last":["foo.A","foo.D"],"up_to_seq":23}`
6. get the last messages for the specified subject(s), where the last message is less than or equal to the up to time, limited by batch size.
    - API: `batch: number, subject: []string, up_to_time: time`

- Request: `{"batch":2,"multi_last":["foo.A","foo.D"],"up_to_time":"2024-11-05T00:50:25.248431300Z"}`

Response Format

Responses may include these status codes:

- `204` indicates the end of a batch of messages, the description header would have value `EOB`
- `404` if the request is valid but no matching message found in stream
- `408` if the request is empty or invalid
- `413` when a multi subject get matches too many subjects

Error code is returned as a header, e.g. `NATS/1.0 408 Bad Request`. Success returned as `NATS/1.0` with no code.

Direct Get replies contain the message along with the following message headers:

- `Nats-Stream`: stream name
- `Nats-Sequence`: message sequence number
- `Nats-Time-Stamp`: message publish timestamp
- `Nats-Subject`: message subject
- `Nats-Num-Pending`: when batched, the number of messages left in the stream matching the batch parameters
- `Nats-Last-Sequence`: when batched, the stream sequence of the previous message
- `Nats-UpTo-Sequence`: when doing multi subject gets the sequence should be used for following requests to ensure consistent reads

A *regular* (not JSON-encoded) NATS message is returned (from the stream store).

Example calls

Direct Get (last_by_subj)

Request:

```
PUB $JS.API.DIRECT.GET.KV_mykv1 _INBOX.6ZtubEqXICZLn7AI4uEiPQ.MmLZadFE 35
{"last_by_subj":"$KV.mykv1.mykey1"}
```

Reply:

```
HMSG _INBOX.6ZtubEqXICZLn7AI4uEiPQ.MmLZadFE 1 143 148
NATS/1.0
Nats-Stream: KV_mykv1
Nats-Subject: $KV.mykv1.mykey1
Nats-Sequence: 1
Nats-Time-Stamp: 2022-08-06 00:29:27.587861324 +0000 UTC

hello
```

Direct Get (next_by_sub starting at seq)

Request:

```
PUB $JS.API.DIRECT.GET.KV_mykv1 _INBOX.pOdFG6hX5uAxqs0JWrAwoY.XJcZuCY4 44
{"seq":1, "next_by_subj":"$KV.mykv1.mykey2"}
```

Reply:

```
HMSG _INBOX.pOdFG6hX5uAxqs0JWrAwoY.XJcZuCY4 1 143 150
NATS/1.0
Nats-Stream: KV_mykv1
Nats-Subject: $KV.mykv1.mykey2
Nats-Sequence: 2
Nats-Time-Stamp: 2022-08-07 06:47:46.610665303 +0000 UTC

goodbye
```

Subject-Appended Direct Get

Request:

```
PUB $JS.API.DIRECT.GET.KV_mykv1.$KV.mykv1.mykey1
_INBOX.qoxA09fQH9fZqNNqrGNPLg.DYPxJr9Y 0`
```

Reply:

```
HMSG _INBOX.qoxA09fQH9fZqNNqrGNPLg.DYPxJr9Y 1 143 148
NATS/1.0
Nats-Stream: KV_mykv1
Nats-Subject: $KV.mykv1.mykey1
Nats-Sequence: 1
Nats-Time-Stamp: 2022-08-06 00:29:27.587861324 +0000 UTC

hello
```

Illegal Subject-Appended Direct Get

Request:

```
PUB $JS.API.DIRECT.GET.KV_mykv1.$KV.mykv1.mykey2
_INBOX.4NogKOPzKEWTqhf4hFIUJV.yo2tE1ep 44
{"seq":1, "next_by_subj":"$KV.mykv1.mykey2"}
```

Reply:

```
HMSG _INBOX.4NogKOPzKEWTqhf4hFIUJV.yo2tE1ep 1 28 28
NATS/1.0 408 Bad Request
```