Pull Consumer Priority Groups

Metadata	Value
Date	2024-05-14
Author	@ripienaar
Status	Approved
Tags	jetstream, server, 2.11

Revision	Date	Author	Info
1	2024-05-14	@ripienaar	Initial design
2	2024-10-15	@jarema	Add client implementation details

Context and Problem Statement

We have a class of feature requests that all come down to adding behaviours on a consumer that affects delivery to groups of clients who interact with the consumer.

Some examples:

- A single client should receive all messages from a consumer and if it fails another should take over
- Groups of related clients should be organised such that certain clients only receive messages when various conditions are met
- Groups of clients should access a Consumer but some should have higher priority than others for receiving messages
- A consumer should deliver messages to clients in a specific grouping such that related messages all go to the same client group

The proposed feature here address some of these needs while paving the way for future needs to be addressed within this framework and under this umbrella feature called `Pull Consumer Groups`.

The current focus is around providing building blocks that can be used to solve higher order problems client-side.

Related prior work:

- Proposed Consumer Groups
- Proposed Partitioned consumer groups and exclusive consumer
- Proposed Consumer Owner ID

General Overview

We introduce 2 settings on `ConsumerConfig` that activates these related features:

```
{
    PriorityGroups: ["<group>", ...],
    PriorityPolicy: "<policy>"
}
```

The presence of the `PriorityPolicy` set to a known policy activates the set of features documented here, `PriorityGroups` require at least one entry.

Technically in some of the proposed policies the `PriorityGroups` have no real meaning today, but we keep it for consistency and allow for future features to be added that would be per-group without then requiring all clients to be updated. Future message grouping features would require groups to be listed here.

In the initial implementation we should limit `PriorityGroups` to one per consumer only and error should one be made with multiple groups. In future iterations multiple groups will be supported along with dynamic partitioning of stream data.

Valid `PriorityGroups` values must match `limited-term` (`(A-Z, a-z, 0-9, dash, underscore, fwd-slash, equals)+`) from ADR-6 and may not exceed 16 characters.

This is only supported on Pull Consumers, configuring this on a Push consumer must raise an error.

> [!NOTE] Some aspects of configuration is updatable, called out in the respective sections, but we cannot support updating a consumer from one with groups to one without and vice versa due to the internal state tracking that is required and the fact that generally clients need adjustment to use these features. We have identified some mitigation approaches to ease migration but will wait for user feedback. We also cannot switch between different policies.

Priority Policies

`overflow` policy

Users want certain clients to only get messages when certain criteria are met.

Imagine jobs are best processed locally in `us-east-1` but at times there might be so many jobs in that region that `us-west-1` region handling some overflow, while less optimal in terms of transit costs and latency, would be desirable to ensure serving client needs as soon as possible.

The `overflow` policy enables Pull requests to be served only if criteria like `num_pending` and `num_ack_pending` are above a certain limit, otherwise those Pull requests will sit idle in the same way that they would if no messages were available (receiving heartbeats etc).

```
{
    PriorityGroups: ["jobs"],
    PriorityPolicy: "overflow",
    AckPolicy: "explicit",
    // ... other consumer options
}
```

Here we state that the Consumer has one group called `jobs`, it is operating on the `overflow` policy and requires `explicit` Acks, any other Ack policy will produce an error. If we force this ack policy in normal use we should error in Pedantic mode.

Pull requests will have the following additional fields:

- `"group": "jobs"` - the group the pull belongs to, pulls not part of a valid group will result in an error
- `"min_pending": 1000` - only deliver messages when `num_pending` for the consumer is >= 1000
- `"min_ack_pending: 1000` - only deliver messages when `ack_pending` for the consumer is >= 1000

If `min_pending` and `min_ack_pending` are both given either being satisfied will result in delivery (boolean OR).

In the specific case where MaxAckPending is 1 and a pull is made using `min_pending: 1` this should only be served when there are no other pulls waiting. This means we have to give priority to pulls without conditions over those with when considering the next pull that will receive a message.

Once multiple groups are supported consumer updates could add and remove groups.

`pinned_client` policy

Users want to have a single client perform all the work for a consumer, but they also want to have a stand-by client that can take over when the primary, aka `pinned` client, fails.

NOTE: We should not describe this in terms of exclusivity as there is no such guarantee, there will be times when one client think it is pinned when it is not because the server switched.

The `pinned_client` policy provides server-side orchestration for the selection of the pinned client.

```
{
    PriorityGroups: ["jobs"],
    PriorityPolicy: "pinned_client",
    PriorityTimeout: 120*time.Second,
    AckPolicy: "explicit",
    // ... other consumer options
}
```

This configuration states:

- We have 1 group defined and all pulls have to belong to this group
- The policy is `pinned_client` that activates these behaviors
- When a pinned client has not done any pulls in the last 120 seconds the server will switch to another client
- AckPolicy has to be `explicit`. If we force this ack policy in normal use we should error in Pedantic mode

A pull request will have the following additional fields:

- `"group": "jobs"` - the group the pull belongs to, pulls not part of a valid group will result in an error

- `"id": "xyz"` - the pinned client will have this ID set to the one the server last supplied (see below), otherwise this field is absent

After selecting a new pinned client, the first message that will be delivered to this client, and all future ones, will include a Nats-Pin-Id: xyz header. The client that gets this message should at that point ensure that all future pull requests have the same ID set.

When a new pinned client needs to be picked - after timeout, admin action, first delivery etc, this process is followed:

1. Stop delivering messages for this group, wait for all in-flight messages to be completed, continue to serve heartbeats
2. Pick the new pinned client
3. Store the new pinned `nuid`
4. Deliver the message to the new pinned client with the ID set
5. Create an advisory that a new pinned client was picked
6. Respond with a 4xx header to any pulls, including waiting ones, that have a different ID set. Client that received this error will clear the ID and pull with no ID

If no pulls from the pinned client is received within `PriorityTimeout` the server will switch again using the same flow as above.

Future iterations of this feature would introduce the concept of a priority field so clients can self-organise but we decided to deliver that in a future iteration.

Clients can expose call-back notifications when they become pinned (first message with `Nats-Pin-Id` header is received) and when they lose the pin (they receive the 4xx error when doing a pull with a old ID).

A new API, `$JS.API.CONSUMER.UNPIN.<STREAM>.<CONSUMER>`, can be called which will clear the ID and trigger a client switch as above. The payload of the message should be the JSON `{"group": "groupName"}`

Consumer state to include a new field `PriorityGroups` of type `[]PriorityGroupState`:

```
type PriorityGroupState struct {
    Group          string `json:"name"`
    PinnedClientId string `json:"pinned_id,omitempty"`
    PinnedTs       *time.Time `json:"pinned_ts,omitempty"`
}
```

Future iterations will include delivery stats per group.

Once multiple groups are supported consumer updates could add and remove groups. Today only the `PriorityTimeout` setting supports being updated.

Advisories

We will publish advisories when a switch is performed and when a pin is lost.

```
const  JSAdvisoryConsumerPinnedPre = "$JS.EVENT.ADVISORY.CONSUMER.PINNED"
const JSAdvisoryConsumerUnpinnedPre = "$JS.EVENT.ADVISORY.CONSUMER.UNPINNED"
const JSConsumerGroupPinnedAdvisoryType =
"io.nats.jetstream.advisory.v1.consumer_group_pinned"

// JSConsumerGroupPinnedAdvisory indicates that a group switched to a new pinned
client
type JSConsumerGroupPinnedAdvisory struct {
   TypedEvent
   Account           string      `json:"account,omitempty"`
   Stream            string      `json:"stream"`
   Consumer          string      `json:"consumer"`
   Domain            string      `json:"domain,omitempty"`
   Group             string      `json:"group"`
   PinnedClientId    string      `json:"pinned_id"`
   Client            *ClientInfo `json:"client"` // if available
}

const JSConsumerGroupUnPinnedAdvisoryType =
"io.nats.jetstream.advisory.v1.consumer_group_unpinned"

// JSConsumerGroupUnPinnedAdvisory indicates that a pin was lost
type JSConsumerGroupUnPinnedAdvisory struct {
   TypedEvent
   Account           string      `json:"account,omitempty"`
   Stream            string      `json:"stream"`
   Consumer          string      `json:"consumer"`
   Domain            string      `json:"domain,omitempty"`
   Group             string      `json:"group"`
   // one of "admin" or "timeout", could be an enum up to the implementor to
decide
   Reason            string      `json:"reason"`
}
```

Client side implementation

Pull Requests changes

To use either of the policies, a client needs to expose a new options on `fetch`, `consume`, or any other method that are used for pull consumers.

Groups

- `group` - mandatory field. If consumer has configured `PriorityGroups`, every Pull Request needs to provide it.

Overflow

When Consumer is in `overflow` mode, user should be able to optionally specify thresholds for pending and ack pending messages.

- `min_pending` - when specified, this Pull request will only receive messages when the consumer has at least this many pending messages.
- `min_ack_pending` - when specified, this Pull request will only receive messages when the consumer has at least this many ack pending messages.

Pinning

In pinning mode, user does not have to provide anything beyond `group`. Client needs to properly handle the `id` sent by the server. That applies only to `Consume`. Fetch should not be supported in this mode. At least initially.

1. When client receives the `id` from the server via `Nats-Pin-Id` header, it needs to store it and use it in every subsequent pull request for this group.
2. If client receives `423` Status error, it should clear the `id` and continue pulling without it.
3. Clients should implement the `unpin` method described in this ADR.