KIP-1017: Health check endpoint for Kafka Connect

Status

Current state: Under discussion

Discussion thread: here

JIRA: here

Please keep the discussion on the mailing list rather than commenting on the wiki (wiki discussions get unwieldy fast).

Motivation

Determining whether a Kafka Connect worker has completed startup and is healthy (or at least, capable of handling REST requests) is surprisingly tricky.

Readiness

In distributed mode, there are several operations a worker completes during startup:

- Creating internal topics if they do not already exist
- Reading to the end of each internal topic
- Joining the cluster (which itself involves participating in a rebalance)

And in standalone mode:

- Creating the offsets file if it did not already exist
- Reading and parsing the offsets file
- Starting all connectors whose configs were specified on the command line
- Generating task configs for all of these connectors
- Starting tasks for all of these connectors

Liveness

In distributed mode, it is also sometimes possible for a worker that has completed startup to still be unhealthy. If any of these operations cannot be performed, the worker will be unable to handle some or all REST requests:

- Remaining in contact with the group coordinator for the cluster
- Reading to the end of the config topic after a rebalance
- If exactly-once support for source connectors is enabled and the worker is the leader of the cluster (see KIP-618: Exactly-Once Support for Source Connectors), instantiating a transactional producer for the config topic
- If session keys are enabled (see KIP-507: Securing Internal Connect REST Endpoints), writing a new session key to the config topic

In standalone, there is one condition that may also make a completely-started-up worker partially or completely unavailable:

- Deadlock

Existing means

There is no one-size-fits-all approach to monitor the readiness and liveness of Kafka Connect workers today, but there is one less-than-elegant option available to cover a subset of the cases.

To check for the **readiness and liveness of a distributed worker or the liveness of a standalone worker**, a request can be issued to the `GET /connectors/{connector}` endpoint. If a response with either a 200 or 404 status code is received, the worker can be considered healthy. This has the drawback of either requiring a connector to exist with the expected name (which may be inconvenient for cluster administrators to enforce), or requiring any automated tooling that interacts with the endpoint to count a 404 response as "healthy", which is highly abnormal. This also does sufficiently confirm the readiness of a standalone worker.

Other approaches, such as querying the `GET /` or `GET /connectors` endpoints, do not actually test for the readiness of either distributed or standalone workers at all. In standalone mode, these requests can be completed before the worker has finished starting all of the command-line-supplied connectors, and in distributed mode, they can be completed before the worker has read to the end of the config topic, joined the cluster, etc.

It's also possible to read worker logs and deduce based on those whether a worker has completed startup, but no sane maintainer of a REST-based project should expect its users to parse log files as a means of checking for node readiness–or at least, to be happy doing so.

Goals

- Establish a single, simple REST endpoint that can be used to check for worker health
- This endpoint should cover both readiness and liveness (i.e., a 2XX response from this endpoint indicates that the worker has both completed startup and is capable of serving other REST requests)
- This endpoint should be available in both standalone and distributed mode, and its semantics should vary as little as possible across the two modes

Public Interfaces

A new REST endpoint, `GET /health` , will be added to the Kafka Connect REST API. If the worker is healthy, the response will have a 200 status code and its body will be a simple JSON object:

GET /health (200)

```
{
  "status": "healthy",
  "message": "Worker has completed startup and is ready to handle requests"
}
```

If the worker has not yet completed startup, the response will have a 503 status code and its body will have a different message:

GET /health (503)

```
{
  "status": "starting",
  "message": "Worker is still starting up"
}
```

If the worker has completed startup but is unable to respond in time, the response will have a 500 status code and its body will have this message:

GET /health (500)

```
{
  "status": "unhealthy",
  "message": "Worker was unable to handle this request and may be unable to handle other requests"
}
```

Unlike other endpoints, the timeout for the health check endpoint will not be 90 seconds. If a consecutive number of N failures reported by this endpoint is required before automated tooling declares the worker unhealthy, then waiting N * 1.5 minutes for an issue with worker health to be detected is likely to be too long. Instead, the timeout for this endpoint will be 10 seconds. In the future, the timeout may be made user-configurable if, for example, KIP-882: Kafka Connect REST API configuration validation timeout improvements or something like it is adopted.

Proposed Changes

Distributed mode

Requests to this endpoint will go through the worker's tick thread. Copying from the current trunk, a comment in that part of the code base explains the purpose of the tick thread:

> *The main loop does two primary things: 1) drive the group membership protocol, responding to rebalance events as they occur, and 2) handle external requests targeted at the leader. All the "real" work of the herder is performed in this thread, which keeps synchronization straightforward at the cost of some operations possibly blocking up this thread (especially those in callbacks due to rebalance events).*

Frequently, bugs and other poor external conditions will block the worker's tick thread, which in turn prevents the worker from being able to handle some REST requests. Affect requests include connector creation/reconfiguration/deletion, reading connector/task configs, restarting connectors/tasks, and reading/resetting/altering connector offsets.

This thread is also responsible for handling worker startup logic.

By handling requests to the health check endpoint on this thread, we can guarantee that the ability to correctly handle a request means that the worker has completed startup, and is capable of handling other REST requests.

Standalone mode

Requests to this endpoint will require synchronization on the `StandaloneHerder` instance, which has been susceptible to deadlock in the past.

They will also only be completed after all command-line-supplied connectors have been successfully created by the herder.

Compatibility, Deprecation, and Migration Plan

This change should be fully backwards compatible. Users who already have their own strategies for monitoring worker health/liveness can continue to employ them. Users who would like to use the endpoint introduced by this KIP need only upgrade to a newer version of Kafka Connect.

Test Plan

Tests should be fairly lightweight. We will cover the three possible states (starting, healthy, unhealthy) in the two different modes (standalone, distributed) through either integration or system testing.

Rejected Alternatives

Reuse existing endpoints

Summary: reuse the existing `GET /` endpoint, or a similar endpoint, as the health check endpoint.

Rejected because:

- It's not clear that users are already employing this endpoint as a health check endpoint, so it's not guaranteed that this would ease the migration path significantly
- Altering the semantics for existing endpoints may have unanticipated effects on existing deployments (users may want to be able to query the version of the worker via the REST API as soon as the worker process has started, without waiting for the worker to complete startup–this may be especially valuable if the worker is unhealthy and that information is not easily accessible otherwise)
- It's easier to add new, user-friendly response bodies that cover the worker state, without worrying about breaking usages of existing endpoints.
- By using a new endpoint for health checks, users can be guaranteed that the endpoint comes with the desired semantics; if an existing endpoint were reused, it would be unclear (without prior knowledge of worker version and Kafka Connect changelog history) whether it would be suitable for health checks

Delay REST endpoint availability until worker has completed startup

Summary: issue 503 responses from all existing endpoints (including `GET /`, `GET /connectors`, `POST /connectors`, etc.) if the worker hasn't completed startup yet, or delay responses from these endpoints until the worker has completed startup (using the existing 90 second timeout)

Rejected because:

- All concerns outlined above for reusing existing endpoints as health check endpoints apply
- This does not cover worker liveness, and only covers worker readiness