KIP-976: Cluster-wide dynamic log adjustment for Kafka Connect

Status

Current state: Accepted

Discussion thread: here

Voting thread: here

JIRA: here

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

Motivation

KIP-495 added a REST API to Kafka Connect that allowed cluster administrators to dynamically adjust log levels at runtime, without having to restart workers or alter their logging configuration file (e.g., the config/connect-log4j.properties file).

While this feature has proven quite valuable over time for debugging some connectors without disrupting the availability of others that were partially or entirely running on the same worker, its one-REST-request-per-worker model can be cumbersome in some scenarios:

- When debugging issues with the Connect runtime itself, especially ones related to cluster membership and rebalances, it can be useful to adjust log levels for every worker in the cluster
- When debugging issues with specific connectors or types of connector, it can be necessary to adjust log levels for a large number of workers (every worker that hosts an instance of that connector or connector type)
- When running Kafka Connect (or at least, its public-facing REST API) behind a load balancer, it may not be possible to target a specific worker with a dynamic log adjustment request

As an incremental improvement to this feature, we can add support for broadcasting dynamic log adjustments to every worker in the cluster.

Public Interfaces

Scope query parameter

A query parameter, `scope`, will be added to the existing `PUT /admin/loggers/{logger}` endpoint. The recognized values (case-insensitive) for this parameter will be:

- **Worker** (default if no value is specified): This will mirror the current behavior of the endpoint: only the worker that receives the request will adjust its log levels, and the response body will include a list of the modified logging namespaces

    - **Cluster**: This will cause every worker in the cluster to adjust its log levels; the response will have no body and its status will be 204 (no content)
    - **Any other value**: This will cause the worker to issue a warning log message, but otherwise handle the request as if no query parameter were specified

Last modified timestamp

The existing `GET /admin/loggers` and `GET /admin/loggers/{logger}` endpoints will be augmented to provide a timestamp for when the logging level for each namespace was last modified. The timestamp will be a standard Unix timestamp with millisecond precision–that is, it will be the number of milliseconds that have elapsed between January 1st, 1970 and when the namespace was modified on the worker. Timestamps will be updated regardless of whether the namespace update was applied using `scope=worker` or `scope=cluster`.

Modification times will be tracked in-memory and determined by when they are applied by the worker, as opposed to when they are requested by the user or persisted to the config topic (details below). If no modifications to the namespace have been made since the worker finished startup, the timestamp will be null.

The `GET /admin/loggers` endpoint will have this new response format, where `${last_modified}` is the last modified timestamp:

GET /admin/loggers

```
[
  ...
  "org.apache.kafka.connect.runtime.WorkerSinkTask": {
    "level": "INFO",
    "last_modified": ${last_modified} // New field
  },
  "org.apache.kafka.connect.runtime.WorkerSourceTask": {
    "level": "DEBUG",
     "last_modified": ${last_modified} // New field
  },
  ...
]
```

The `GET /admin/loggers/{logger}` endpoint will have this new response format (using `${last_modified}` in the same manner as above):

GET /admin/loggers/{logger}

```
{
   "level": "INFO",
   "last_modified": ${last_modified} // New field
 }
```

Proposed Changes

Distributed mode

Servicing REST requests

If the `scope` query parameter is set to `cluster`, the worker that receives this request will write a record to the config topic instructing all workers in the cluster to adjust their log levels. It will then read to the end of the config topic, guaranteeing that at least the worker that received the request has adjusted its log levels. This is similar to how the API to restart all tasks for a connector en masse was implemented as part of KIP-745.

Since cluster metadata is not required to handle these types of request, they will not be forwarded to the leader, and they will be eligible for handling even during rebalances. This is similar (though not identical) to existing logic for pausing and resuming connectors in distributed mode.

Config topic records

Record keys will have the format `"logger-cluster-${logger}"` , where `${logger}` is the logging namespace to adjust.

Record values will have the following format, where `${level}` is the new logging level for the namespace:

Config topic record value format

```
{
  "level": "${level}"
}
```

As an example, when handling a request to set the logging level of the namespace `org.apache.kafka.connect.runtime.distributed.DistributedHerder` to `TRACE` with a `scope` of `cluster` (if you're debugging the distributed herder, you need all the help you can get), a worker would write a record with a key of `"logger-cluster-org.apache.kafka.connect.runtime.distributed.DistributedHerder"` and the following value to the config topic:

Config topic record value example

```
{
  "level": "TRACE"
}
```

When a worker that has completed startup reads one of these records from the config topic, it will apply the requested logging changes in the exact same manner as if they were requested via the existing `PUT /admin/loggers/{logger}` endpoint.

Workers that have not yet completed startup will ignore these records, including if the worker reads one during the read-to-end of the config topic that all workers perform during startup. Restarting a worker will cause it to discard all cluster-wide dynamic log level adjustments, and revert to the levels specified in its Log4j configuration. This mirrors the current behavior with per-worker dynamic log level adjustments.

There may be some delay between when a REST request with `scope=cluster` is received and when all workers have read the corresponding record from the config topic. The last modified timestamp (details above) can serve as a rudimentary tool to provide insight into the propagation of a cluster-wide log level adjustment.

Standalone mode

Given that standalone mode by definition only supports one worker, this feature does not seem applicable on the surface. And, for the underlying dynamic log adjustment logic, no changes will be made. However, for the sake of consistency with distributed mode, the `scope` query parameter will still be recognized and, if set to `cluster`, will cause a 204 response with no body to be returned.

Compatibility, Deprecation, and Migration Plan

Setting logging levels

Existing behavior is preserved as the default for this API. The proposed feature is only available in an opt-in basis.

Getting logging levels

By adding the new `last_modified` field to the response format for these endpoints, we introduce some risk of breaking existing tooling that works with the Kafka Connect REST API. If strict deserialization of JSON responses is performed by these tools, then the new field (which will be unrecognized) will cause failures. These tools will need to be updated to either ignore unrecognized fields, or account for the new field.

Worker downgrades

If a worker is downgraded to an earlier version of Kafka Connect that does not recognize dynamic log adjustment records in the config topic, it will log an error message in response to reading a record from that topic with an invalid key. There will be no other impact (for example, the worker won't fail and the availability of its REST API and the connectors/tasks it's assigned will not be compromised).

Test Plan

Unit tests

- Ensure that records produced to the config topic have the expected format
- Ensure that updates to a logging level are reported with the correct last modified timestamp
- Ensure that logging levels that have not been updated have a null last modified timestamp
- Ensure that distributed workers that have completed startup correctly handle logging adjustment config topic records
- Ensure that distributed workers that have not completed startup ignore logging adjustment config topic records

- Ensure that requests to the existing `PUT /admin/loggers/{logger}` endpoint with no `scope` query parameter, and with `scope=worker` result in the same herder-level behavior as before (mostly likely accomplished by verifying that no interactions with the `Herder` object have taken place)
- Ensure that requests to the existing `PUT /admin/loggers/{logger}` endpoint with an unrecognized value for the `scope` query parameter result in the same herder-level behavior as before, but also cause a warning log message to be emitted
- Ensure that cluster-scoped requests with invalid logging levels are rejected with a 404 response
- Ensure that repeated requests to set the same logging level for a namespace do not cause its last modified timestamp to be updated

Integration tests

A new integration test will be added for standalone mode, which will run through this series of scenarios and assertions:

1. Start a standalone Connect worker
   a. Ensure that the last modified timestamp for all reported logging namespaces is null
2. Modify the logging level for a specific namespace with no `scope` parameter
   a. Ensure that the response body is non-empty and matches the same format it had prior to this KIP
   b. Ensure that the last modified timestamp for that namespace is non-null and at least as recent as the time at which the request was issued
   c. Ensure that the logging level for that namespace is correct
   d. Ensure that the last modified timestamp for all other namespaces is still null
   e. Ensure that no other namespaces have been modified
3. Modify the logging level for a specific namespace with `scope=worker`
   a. Ensure that the response body is non-empty and matches the same format it had prior to this KIP
   b. Ensure that the last modified timestamp for that namespace is non-null and at least as recent as the time at which the request was issued
   c. Ensure that the logging level for that namespace is correct
   d. Ensure that the last modified timestamp for all other namespaces is still null
   e. Ensure that no other namespaces have been modified
4. Issue a second request to set the same logging level for the same namespace with `scope=worker`
   a. Ensure that the last modified timestamp for that namespace is not updated
5. Modify the logging level for a different namespace with `scope=cluster`
   a. Ensure that the response body is empty
   b. Ensure that the last modified timestamp and level for that namespace are correct
   c. Ensure that the last modified timestamp and level for all other namespaces remain unchanged

System tests

A single test will be added that runs through this series of scenarios and assertions:

1. Start a distributed Connect cluster with three workers
   a. Ensure that the last modified timestamp for all reported logging namespaces is null
2. Modify the logging level for a specific namespace for single worker
   a. Ensure that the response body is non-empty and matches the same format it had prior to this KIP
   b. Ensure that the last modified timestamp for that namespace on the affected worker is non-null and at least as recent as the time at which the request was issued (some margin of error may be necessary in the highly unlikely but technically possible event that the node responsible for running tests and the one running the worker have skewed clocks)
   c. Ensure that the logging level for that namespace on the affected worker is reported (via the admin REST API) with the correct level
   d. Ensure that the last modified timestamp for that namespace on all other workers is still null
   e. Ensure that the logging level for that namespace on all other workers is unchanged
3. Modify the logging level for the root namespace for all workers (using `scope=cluster`)
   a. Ensure that the response body is empty
   b. Ensure that, after a reasonable timeout, the logging level for all reported namespaces on all workers is reported with the correct level
   c. Ensure that the last modified timestamp for all namespaces on all workers is non-null and at least as recent as the time at which the request was issued
4. Modify the logging level for a specific namespace for all workers (using `scope=cluster`)
   a. Ensure that the response body is empty
   b. Ensure that, after a reasonable timeout, the logging level for that namespace on all workers is reported with the correct level
   c. Ensure that the last modified timestamp for that namespace on all workers is non-null and at least as recent as the time at which the request was issued
5. Issue a second request to set the same logging level for the same namespace for all workers (using `scope=cluster`)
   a. No assertions will be made for this step
6. Modify the logging level for a different specific namespace for all workers (using `scope=cluster`)
   a. Ensure that, after a reasonable timeout, the logging level for that namespace on all workers is reported with the correct level
   b. Ensure that the last modified timestamp for that namespace on all workers is non-null and at least as recent as the time at which the request was issued
   c. Ensure that the last modified timestamp for the namespace affected in steps 4 and 5 is unchanged from when it was tested in step 4 (i.e., the second request in step 5 did not affect it)
7. Modify the logging level for the root namespace for all workers (using `scope=cluster`)
   a. No assertions will be made for this step
8. Modify the logging level for a specific namespace for a single worker (again)
   a. Ensure that the response body is non-empty and matches the same format it had prior to this KIP
   b. Ensure that the last modified timestamp for that namespace on the affected worker is at least as recent as the time at which the request was issued
   c. Ensure that the logging level for that namespace on the affected worker is reported with the correct level
   d. Ensure that the last modified timestamp for all namespaces except the modified namespace on the affected worker, and all namespaces for all other workers, is unchanged since the root level was modified for all workers*****
   e. Ensure that the logging levels for all namespaces except the modified namespace on the affected worker, and all namespaces for all other workers, is unchanged since the root level was modified for all workers*****

***** - Note that assertions like these ("ensure that <condition> is not met") are difficult to test for; if there is a bug in the logic under test that causes the condition to eventually be met, but after the point where it is observed, then these tests are liable to report spurious successes. We rely on unit testing coverage to prevent the kinds of bugs that would cause these spurious successes, as opposed to, e.g., sleeping for <n> seconds before checking the condition.

No efforts will be made to verify the actual contents of the logs for any workers. KIP-495 was published several years ago and has proven to be effective; since we anticipate that the logic for reading/writing log levels will be largely preserved, it should be enough to rely on the API for querying the Kafka Connect-reported levels of logging namespaces.

A system test is used here instead of one or more integration tests because the latter colocate workers with the same JVM, making it difficult to distinguish between changes to the logging levels of a single worker and the whole cluster.

Rejected Alternatives

Request-time modified timestamp tracking

Instead of tracking the last modified timestamp for a logging namespace based on when it was applied by a worker, we could track it by when the request was received, or when it was written to the config topic. This would provide at least one advantage: assuming all workers are caught up on the config topic, every worker would give the exact same response for requests to view the levels of loggers. However, it would also be less accurate: users may be dismayed to see that the logging level for a given namespace had a last modified time of *T*, but that the actual level of logs emitted by that worker for that namespace was different until time *T+n*, for some non-negative number *n*.

Versioned request format

In order to work better with tools that use strict deserialization, we could add either opt-out or opt-in logic to receive requests from endpoints that provide levels for logging namespaces with the newly-proposed format (i.e., with the last modified timestamp). This could come, for example, in the form of a new request header that dictates which version of the API that clients expect.

This change may be smoother for users, but would come with some significant costs:

- Higher maintenance burden: we would have to be able to serve requests that expect both kinds of response format
- Setting an expensive precedent for the Kafka Connect REST API: unless absolutely necessary, we should encourage consumers of the API to tolerate unknown fields in order to permit flexibility in future changes we may opt to make that would only involve adding new fields

Persistent logging level updates

Both the new cluster-wide API proposed in this KIP and the existing worker-local API added in KIP-495 only support ephemeral updates: any dynamic logging level changes will be discarded if a worker restarts, and the worker will revert to the levels specified in its Log4j configuration.

The rationale for keeping these updates ephemeral is to continue to give priority to workers' Log4j configuration files, with the underlying philosophy that this endpoint is still only intended for debugging purposes, as opposed to cluster-wide configuration. Permanent changes can already be made by tweaking the Log4j file for a worker and then restarting it. If a restart is too expensive for a permanent change, then the change can be applied immediately via the REST API, and staged via the Log4j configuration file (which will then be used the next time the worker is restarted, whenever that happens).

Future work

More scope types

We may want to add more fine-grained scopes for adjusting log levels. For example, adjusting the log levels of all workers that are running a `Connector` and/or `Task` for a specific connector, or all workers that are running a `Connector` and/or `Task` for a specific connector type. This could be accomplished with additional values for the `scope` parameter such as `scope=connector:reddit-comments` or `scope=connectorType:BigQuerySink` or `scope=taskType:io.debezium.connector.mysql.MySqlConnectorTask`. Or, new query parameters could be added such as `connector=reddit-comments`, etc.

It's likely that this scoping information would have to be embedded in the keys of records written to the config topic, to avoid accidentally compacting other records whose scopes differ.

Enable by default

If this feature is popular enough, we may consider changing the default of the `scope` parameter from `worker` to `cluster`, since this would arguably be more convenient and intuitive for users of Kafka Connect.

Config topic cleanup logic

This proposal introduces a second kind of record to the config topic that's used for cluster-wide communication, and is meant to be ignored by any workers brought up after it has been written (the first kind being the one added in KIP-745). These kinds of records runs the risk of flooding the config topic with many records that, due to the compacted nature of the topic, will never be discarded, leading to a monotonically-growing topic.

Practically speaking, it's unclear that this will be an issue. Adjusting logging levels is an incredibly useful feature, but the value it provides is most applicable when human beings (not automated tools) are debugging unusual circumstances. It's highly unlikely that users or tools will be issuing so many dynamic log level adjustment requests that the config topic grows to an unmanageable size.

However, we may still want to invest some time in cleanup logic for the config topic, where records like the ones proposed here and introduced in KIP-745 are followed up with corresponding tombstone records, so that when compaction takes place, they are effectively removed from the topic. These tombstones could possibly be emitted after a fixed delay has elapsed, or possibly after a rebalance has taken place (since every worker reports its current offset in the config topic).