KIP-475: New Metrics to Measure Number of Tasks on a Connector

- Status
- Motivation
- Public Interfaces
- Proposed Changes
  - W1 reports:
  - W2 reports:
- Compatibility, Deprecation, and Migration Plan
- Rejected Alternatives
  - Connector task counts on the connector level
  - Specify Worker ID as a Tag
  - Excluding the "connector-" prefix
  - Rely on the REST API

Status

Current state: Accepted

Discussion thread: here

Voting thread: here

JIRA: here

PR: here

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

Motivation

Connect framework provides good metrics for monitoring behavior. Currently, we have a metric on the Worker to measure number of tasks.

It is useful in many applications to measure the number of tasks on a Connector. For example, an administrator may wonder the average number of tasks per connector, which types of connectors create the most tasks, if some connectors are outliers (creating too many or too few tasks). The current metric providing tasks per Worker is useful in its own right, but does not address any of these needs.

Further, it is useful to know, within a worker, how many of a connector's tasks are allocated to different workers. This allows us to answer questions like "Are my connector's tasks well-balanced?"

It is also useful to know the breakdown of how many tasks on a connector are running, paused, or failed.

Public Interfaces

We propose adding the following new metrics on the existing group name "connect-worker-metrics" in the ConnectMetricsRegistry. Since each process represents a Worker, including the task count metrics scoped by the "connector=()" tag is sufficient to

MBean	Metric/Attribute Name	Description
`kafka.connect:type=connect-worker-metrics, connector=([-.\w]+)`	connector-total-task-count	The number of tasks of the connector on the worker.
`kafka.connect:type=connect-worker-metrics, connector=([-.\w]+)`	connector-running-task-count	The number of running tasks of the connector on the worker.
`kafka.connect:type=connect-worker-metrics, connector=([-.\w]+)`	connector-paused-task-count	The number of paused tasks of the connector on the worker.
`kafka.connect:type=connect-worker-metrics, connector=([-.\w]+)`	connector-failed-task-count	The number of failed tasks of the connector on the worker.
`kafka.connect:type=connect-worker-metrics, connector=([-.\w]+)`	connector-unassigned-task-count	The number of unassigned tasks of the connector on the worker.
`kafka.connect:type=connect-worker-metrics, connector=([-.\w]+)`	connector-destroyed-task-count	The number of destroyed tasks of the connector on the worker.

Since each task must always have exactly one non-null status, and we've covered every task status, the "connector-total-task-count" will be equal to the sum of each status.

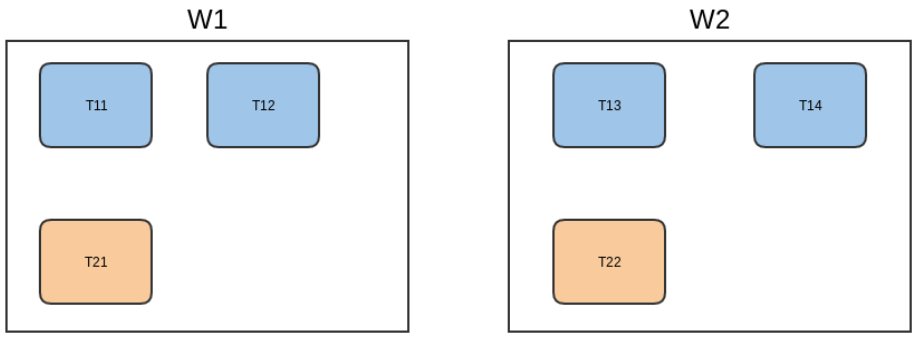
Proposed Changes

The above metrics will be added. These metric will be calculated within the Worker.WorkerMetricsGroup's constructor, which already has access to the worker's tasks (how the existing "task-count" metric is calculated). These tasks will be further broken down by connector and task status to implement this KIP.

Metrics gathering will perform efficiently and be cheap to calculate. The newly proposed metrics are calculated in-memory on existing objects, and calculations can be O(n) on the number of tasks on the metric, which is the best you can do.

An example is shown below of the metrics if we have

- Two workers, W1 and W2
- Two Connectors: C1 is owned by W1 and C2 is owned by W2
- C1 (blue in the diagram) has tasks T11, T12 on W1 and T13 and T14 on W3
- C2 (orange in the diagram) has tasks T21 on W1 and T22 on W2



Then the metrics will be:

W1 reports:

MBean	Metric Name	Value
`kafka.connect:type=connect-worker-metrics,connector=C1`	connector-total-task-count	2
`kafka.connect:type=connect-worker-metrics,connector=C2`	connector-total-task-count	1

and similar values for each of the "running", "paused", "failed", "unassigned", and "destroyed" statuses.

W2 reports:

Bean	Metric Name	Value
`kafka.connect:type=connect-worker-metrics,connector=C1`	connector-total-task-count	2
`kafka.connect:type=connect-worker-metrics,connector=C2`	connector-total-task-count	1

and similar values for each of the "running", "paused", "failed", "unassigned", and "destroyed" statuses.

Thus, if an analyst wants to know the total number of tasks for C1, they will add the value for the appropriate metric for each worker, and find C1 has 4 tasks. Similarly, they will find C2 has 2 tasks.

If an analyst wishes to confirm that C2 is well-balanced, they will gather the the value for "connector-total-task-count" for each worker, and plot a histogram, and see that it is evenly distributed.

Compatibility, Deprecation, and Migration Plan

This KIP simply adds new metrics.

Rejected Alternatives

Connector task counts on the connector level

Using an MBean like "kafka.connect:type=connector-metrics,connector=([-.\w]+)" would give us per-connector metrics. The drawback to this strategy is that we are forgoing some information: We don't know which worker has how many of each connector's tasks. This strategy also forces us to give the WorkerConnector reference to a Herder in order to calculate the task count. It is an OK solution, but the proposed solution above gives more useful data, and is cleaner to implement.

Specify Worker ID as a Tag

This is rejected since each process is a worker, so worker ID can be inferred from the process. Furthermore, the proposed design keeps with the established pattern of how worker tasks are exposed.

Excluding the "connector-" prefix

"connector-" prefix keeps naming consistent with the existing names "connector-version", "connector-type", and "connector-status". Further, while not strictly necessary, it does disambiguate from the Worker's "task-count" metric name.

Rely on the REST API

While the Connect REST API does provide the number of tasks for a connector, it is unuseful in Standalone Mode. Further, exposing this number through the common metrics interface enables downstream interfaces and pipelines to gather metrics from a single source.