KIP-875: First-class offsets support in Kafka Connect

Status

Current state: *Accepted*

Discussion thread: here

Vote thread: here

JIRA: here

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

Motivation

Since its creation, the Kafka Connect Java API has provided support for offsets, which track the progress of both source and sink connectors. Tasks can use these offsets on startup to begin reading from the last-consumed point in their upstream system (i.e., Kafka for sink tasks and the external system for source tasks), instead of starting over at the beginning (causing duplicate data to be written) or at the end (causing data to be missed).

However, Kafka Connect does not yet provide an official API for cluster administrators to view or modify the offsets of connectors on the cluster. Some cases where this may be useful include:

- Resetting the offsets for a connector while iterating rapidly in a development environment (so that the connector does not have to be renamed each time)
- Viewing the in-memory offsets for source connectors on the cluster in order to recover from accidental deletion of the Connect source offsets topic (this is currently possible with the config topic, and can be a life-saver)
- Monitoring the progress of running connectors on a per-source-partition basis
- Skipping records that cause issues with a connector and cannot be addressed using existing error-handling features

Kafka Connect has also evolved over the years to support some use cases that make offset management complex. These include:

- Creating connectors that target a different Kafka cluster than the one that hosts the Kafka Connect cluster's internal topics (possible as of KIP-458: Connector Client Config Override Policy)
- Creating sink connectors with an overridden, user-specified consumer group ID (possible as of KIP-458: Connector Client Config Override Policy)
- Creating source connectors with a custom offsets topic (possible as of KIP-618: Exactly-Once Support for Source Connectors)

Any support for viewing and manipulating connector offsets added to Kafka Connect should accommodate all of these cases.

Public Interfaces

Endpoints

Several new endpoints will be added to the Kafka Connect REST API:

Verb	Path	Summary
GET	/connectors/{connector} /offsets	Retrieve the offsets for a connector; the connector must exist
PATCH	/connectors/{connector} /offsets	Alter the offsets for a connector; the connector must exist, and must be in the STOPPED state (described below)
DELETE	/connectors/{connector} /offsets	Reset the offsets for a connector; the connector must exist, and must be in the STOPPED state (described below)
PUT	/connectors/{connector}/stop	Stop the connector, but do not delete it (described below); the connector must exist

Request/response format

Reading offsets (response)

Queries for a connector's offsets will use one of these formats, depending on whether the offsets are for a source or a sink connector. Care is taken to keep the two formats symmetrical.

Source offsets response

```
{
  "offsets": [
    {
      "partition": {
        // Connector-defined source partition
          },
      "offset": {
        // Connector-defined source offset
      }
    }
  ]
}
```

Sink offsets response

```
{
  "offsets": [
    {
      "partition": {
        "kafka_topic": // Kafka topic
        "kafka_partition": // Kafka partition
          },
      "offset": {
        "kafka_offset": // Kafka offset
      }
    }
  ]
}
```

Altering offsets (request)

Requests to alter a connector's offsets will use an identical body format. This will allow users to reuse the responses from the offset-read endpoint in the bodies of requests to alter offsets.

Source offsets request

```
{
  "offsets": [
    {
      "partition": {
        // Connector-defined source partition
          },
      "offset": {
        // Connector-defined source offset
      }
    }
  ]
}
```

Sink offsets request

```
{
  "offsets": [
    {
      "partition": {
        "kafka_topic": // Kafka topic
        "kafka_partition": // Kafka partition
          },
      "offset": {
        "kafka_offset": // Kafka offset
      }
    }
  ]
}
```

For either kind of connector, the offset field may be `null` , which will reset the offset for that specific partition. For example, to reset the offset for partition 3 of a topic T read by a sink connector:

Sink offsets request (patch reset)

```
{
  "offsets": [
    {
      "partition": {
        "kafka_topic": "T"
        "kafka_partition": 3
          },
      "offset": null
    }
  ]
}
```

Altering/resetting offsets (response)

We will want to accurately communicate to users whether they can be certain that offsets have been reset for their connector, which is not possible unless the connector has communicated this information to the Connect runtime by implementing the `alterOffsets` method (described below).

As a result, there are three possible cases we will want to address with the responses for requests to alter or reset offsets.

If the connector has implemented `alterOffsets` and everything has succeeded, the HTTP status will be 200 and the response body will be:

Offset alter/reset response (definite success)

```
{
  "message": "The offsets for this connector have been <reset|altered> successfully"
}
```

If the connector has not implemented `alterOffsets` but everything else has succeeded, the HTTP status will be 200 and the response body will be:

Offset alter/reset response (possible success)

```
{
  "message": "The framework-managed offsets for this connector have been <reset|altered> successfully. However,
if this connector manages offsets externally, they will need to be manually <altered|reset> in the system that
the connector uses."
}
```

If anything fails (including consumer group deletion for sink connectors, zombie fencing for exactly-once source connectors, invoking `alterOffsets` on a connector, etc.), the standard 500 response will be returned:

Offset alter/reset response (possible success)

```
{
  "error_code": 500,
  "message": // Exception message here
}
```

Endpoints behavior

Reading offsets

The `GET /connectors/{connector}/offsets` endpoint will be useful for examining the offsets of a currently-running connector. It will automatically deduce whether the connector is a source or sink based on the connector's configuration, and then return the offsets for the connector. The request will fail with a 404 response if the connector does not exist on the cluster.

Altering offsets

The `PATCH /connectors/{connector}/offsets` endpoint will be useful for altering the offsets of stopped connectors. Requests will be rejected if the connector does not exist on the cluster (based on the config topic), if a rebalance is pending, if the connector is not in the `STOPPED` state (described below), or if it does not have an empty set of tasks in the config topic. If successful, the request will be met with a 200 response and the appropriate body. The request will fail with a 404 response if the connector does not exist on the cluster, and will fail with a 400 response if the connector does exist but is not in the correct state.

The connector's `alterOffsets` method (described below) will be invoked, before modifying its consumer group offsets or source offsets.

If exactly-once source support is enabled on a worker that receives a request to alter offsets for a source connector, it will fence out all previously-running tasks for the connector (if any exist, based on the presence of a task count record in the config topic) before invoking `alterOffsets` and altering the offsets for the connector. Offsets will be altered transactionally for the connector's primary offsets topic (i.e., the one that is written to transactionally by source tasks when the connector is running). The transactional ID used for this operation will be `${groupId}-${connector}`, where `${groupId}` is the group ID of the Connect cluster and `${connector}` is the name of the connector.

Requests to this endpoint will only augment any existing offsets for the connector; they will not implicitly remove any offsets for the connector that are not included in the request body.

All offset alter requests will be forwarded to the leader of the cluster.

Resetting offsets

The `DELETE /connectors/connector/offsets` endpoint will be useful for resetting the offsets of stopped connectors. Requests will be rejected if the connector does not exist on the cluster (based on the config topic), if a rebalance is pending, if the connector is not in the `STOPPED` state (described below), or if it does not have an empty set of tasks in the config topic. If successful, the request will be met with a 200 response and the appropriate body. The request will fail with a 404 response if the connector does not exist on the cluster, and will fail with a 400 response if the connector does exist but is not in the correct state. This endpoint will be idempotent; multiple consecutive requests to reset offsets for the same connector with no new offsets produced in between those requests will all result in a 200 response (if they are successful).

The connector's `alterOffsets` method (described below) will be invoked, before deleting its consumer group or source offsets. The `offsets` argument will contain keys for every known topic partition in the consumer group (if the connector is a sink) or every known source partition (if the connector is a source), each with a null value.

A source offset will only be considered successfully deleted if the Connect worker is able to emit a tombstone to the offsets topic for its partition, and then read to the end of the offsets topic. A request to reset offsets for a source connector will only be considered successful if the worker is able to delete all known offsets for that connector, on both the worker's global offsets topic and (if one is used) the connector's dedicated offsets topic.

If exactly-once source support is enabled on a worker that receives a request to reset offsets for a source connector, it will fence out all previously-running tasks for the connector (if any exist, based on the presence of a task count record in the config topic) before invoking `alterOffsets` and resetting the offsets for the connector. Offsets will be reset transactionally for the connector's primary offsets topic (i.e., the one that is written to transactionally by source tasks when the connector is running). The transactional ID for this operation will be `${groupId}-${connector}`, where `${groupId}` is the group ID of the Connect cluster and `${connector}` is the name of the connector. If this topic is different from the worker's global offsets topic, the connector's offsets will be removed from that topic as well, but without the use of a transaction.

Requests to reset sink connector offsets will be satisfied by deleting the consumer group for that sink connector (as opposed to deleting all known offsets for that consumer group). Requests to reset sink connector offsets will fail if there are any active members of the sink connector's consumer group.

All offset reset requests will be forwarded to the leader of the cluster.

New target state: STOPPED

Background: target states today

Kafka Connect currently supports two "target states" for a connector: `RUNNING` (the default), and `PAUSED`. The target state for a connector can be controlled using the REST API, using the `PUT /connectors/connector/resume` endpoint for the `RUNNING` state and the `PUT /connectors/connector/pause` endpoint for the `PAUSED` state.

When a connector is paused, its tasks continue to exist on the cluster. Many resources allocated for them, including Kafka and Kafka Connect resources such as clients, converters, and transformations, and connector-specific resources such as database connections, file descriptors, memory buffers, remain allocated. This can lead to confusing and sometimes suboptimal behavior when users pause connectors but notice that resources (especially database connections) are still allocated.

In addition, connectors and tasks are allocated across the cluster without regard to their target state during rebalance. This can lead to some skew in resource usage (such as network traffic and CPU utilization) across the cluster; in the worst case, all paused tasks are allocated to one set of workers, and all running tasks are allocated to a disjoint set of workers.

A new STOPPED state

A new target state will be added for connectors: `STOPPED`. The semantics for a connector that becomes stopped will be:

- The connector config remains present in the config topic of the cluster (if running in distributed mode), unmodified
- The connector config remains visible in the REST API
- All tasks for the connector are shut down completely. If running in distributed mode, a set of empty tasks is published to the config topic for the connector
- If running in distributed mode, as a result of the empty set of task configs being published to the config topic, a rebalance will be triggered, and no tasks for the connector will be allocated across the cluster
- As a further result of that rebalance, any information on the connector provided by the REST API will show it with an empty set of tasks
- The `Connector`, if running, will be shut down (by invoking `Connector::stop` and deallocating all Kafka Connect- and Kafka-related resources allocated for it by Kafka Connect)
- The `Connector` will still appear in the status portion of the REST API, with a state of `STOPPED`. This will take place even if the connector was in the `FAILED` state before the request to stop it, or if it failed during shutdown in response to a request to stop
- If running in distributed mode, the `Connector` will still be assigned to a worker during each rebalance
- The `Connector` will not be started (by invoking `Connector::start`), and it will not be able to generate new task configurations (by invoking `ConnectorContext::requestTaskReconfiguration`)

When a stopped connector is resumed or paused (there is no difference between the two transitions), the semantics will be:

- The `Connector` is started on the worker that it is assigned to, and allowed to generate a set of task configurations
- In standalone mode, tasks are immediately brought up based on that set of configurations
- In distributed mode, if that set of configurations is non-empty, it is written to the config topic, and a rebalance ensues
- All further behavior is identical to any other cases where a new set of task configurations for a connector is generated (including a connector being reconfigured via the REST API, or a connector automatically generating a new set of configurations based on a monitoring thread that polls the external system for changes)

Connector API changes

Hooks will be added to the connector APIs to allow connectors to implement custom logic for offset alterations/resets. These hooks will serve a few different purposes:

- Connectors that manage offsets externally will be able to reject the alter/reset request if their offsets cannot be modified (which may the case if, for example, offset information is tightly coupled with data written to a sink system; developers will be discouraged from implementing offset alter/reset logic that would require clearing data written to a sink system)
- Connectors that manage offsets externally will be able to propagate the alter/reset to the external system they use for offset management if this can be supported
- Connectors will be able to validate user-provided offsets and handle invalid data (this use case is mostly targeted toward source connectors; the offset format for sink connectors will be validated automatically by the Connect runtime)

SourceConnector offset hook

```
public abstract class SourceConnector extends Connector {

    /**
     * Invoked when users request to manually alter/reset the offsets for this connector via the REST
     * API. Connectors that manage offsets externally can propagate offset changes to their external
     * system in this method. Connectors may also validate these offsets if, for example, the source
     * partition is in a format that will not be recognized by them or their tasks.
     * <p/>
     * Connectors that do not manage offsets externally or require custom offset validation need not
     * implement this method beyond simply returning {@code true}.
     * <p/>
     * User requests to alter/reset offsets will be handled by the Connect runtime and will be reflected
     * in the offsets returned by any
     * {@link org.apache.kafka.connect.storage.OffsetStorageReader OffsetStorageReader instances}
     * provided to this connector and its tasks.
     * <p/>
     * Similar to {@link #validate(Map) validate}, this method may be called by the runtime before the
     * {@link #start(Map) start} method is invoked.
     *
     * @param connectorConfig the configuration of the connector
     * @param offsets a map from source partition to source offset, containing the offsets that the
     *                user has requested to alter/reset. For any source partitions that are being reset instead
     *                of altered, their corresponding value in the map will be {@code null}.
     * @return whether this method has been overridden by the connector; the default implementation returns
     * {@code false}, and all other implementations (that do not unconditionally throw exceptions) should return
     * {@code true}
     * @throws UnsupportedOperationException if it is impossible to alter/reset the offsets for this connector
     * @throws org.apache.kafka.connect.errors.ConnectException if the offsets for this connector cannot be
     * reset for any other reason (for example, they have failed custom validation logic specific to this
connector)
     */
    public boolean alterOffsets(Map<String, String> connectorConfig, Map<Map<String, ?>, Map<String, ?>>
offsets) {
        return false;
    }

}
```

SinkConnector offset hook

```
public abstract class SinkConnector extends Connector {

    /**
     * Invoked when users request to manually alter/reset the offsets for this connector via the REST
     * API. Connectors that manage offsets externally can propagate offset changes to their external
     * system in this method. Connectors may also validate these offsets if, for example, an offset
     * is out of range for what can be feasibly written to the external system.
     * <p/>
     * Connectors that do not manage offsets externally or require custom offset validation need not
     * implement this method beyond simply returning {@code true}.
     * <p/>
     * User requests to alter/reset offsets will be handled by the Connect runtime and will be reflected
     * in the offsets for this connector's consumer group.
     * <p/>
     * Similar to {@link #validate(Map) validate}, this method may be called by the runtime before the
     * {@link #start(Map) start} method is invoked.
     *
     * @param connectorConfig the configuration of the connector
     * @param offsets a map from topic partition to offset, containing the offsets that the
     *                user has requested to alter/reset. For any topic partitions that are being reset instead
     *                of altered, their corresponding value in the map will be {@code null}.
     * @return whether this method has been overridden by the connector; the default implementation returns
     * {@code false}, and all other implementations (that do not unconditionally throw exceptions) should return
     * {@code true}
     * @throws UnsupportedOperationException if it is impossible to alter/reset the offsets for this connector
     * @throws org.apache.kafka.connect.errors.ConnectException if the offsets for this connector cannot be
     * reset for any other reason (for example, they have failed custom validation logic specific to this
connector)
     */
    public boolean alterOffsets(Map<String, String> connectorConfig, Map<TopicPartition, Long> offsets) {
        return false;
    }

}
```

Compatibility, Deprecation, and Migration Plan

Offsets endpoints

This feature is fully backwards-compatible with existing Kafka Connect releases. Migration should occur automatically whenever a Kafka Connect cluster is upgraded to a version that supports this feature.

Additional connector ACLs

If exactly-once source support is enabled on a worker, in order to handle requests to alter or reset offsets, the connector's principal must be able to use a transactional ID of `${groupId}-${connector}`, where `${groupId}` is the group ID of the Connect cluster and `${connector}` is the name of the connector.

STOPPED target state

Diligent readers will note that the addition of a new target state creates problems with cluster downgrades and rolling upgrades. If a connector is stopped, and a worker running an older version of Kafka Connect either joins or currently exists in the cluster, that worker may not know how to handle the new record in the config topic that includes the request to stop the connector.

With some careful work, we can actually gracefully degrade in this scenario instead of confusing the worker running the older version of Kafka Connect.

Instead of publishing a "naive" record to the config topic with contents like this:

Key	Value

`target-state-{connector}`	**"Naive" STOPPED record value** `{"state": "STOPPED"}`

We can "fool" older workers into treating `STOPPED` requests as `PAUSE` requests by emitting records with this format:

Key	Value

`target-state-{connector}`	**"Clever" STOPPED record value**
	`{`
	`"state": "PAUSED",`
	`"state.v2": "STOPPED"`
	`}`

Older workers will inspect the `state` field of the record value, see that it is `PAUSED`, and pause the parts of the connector that they are assigned.

Newer workers will first inspect the `state.v2` field and, if it is found, use that as the new target state. If no `state.v2` field is found, they will fall back on the `state` field.

Implementation plan

This KIP proposes several new features, which can be worked on and released independently. If this KIP is accepted, each of these features will have its own JIRA ticket and can be implemented in its own separate PR.

STOPPED state

Scope: The new `PUT /connectors/{connector}/stop` endpoint

Dependencies: None

Reading offsets

Scope: The new `GET /connectors/{connector}/offsets` endpoint

Dependencies: None

Writing offsets

Scope: The new `PATCH /connectors/{connector}/offsets` endpoint

Dependencies: The **STOPPED state** and **Resetting offsets (internal logic)** features

Resetting offsets (internal logic)

Scope: All internal logic necessary to reset the offsets of a connector, given the connector config. Does not include any user-facing interfaces for triggering offset resets.

Dependencies: None

Resetting offsets (stopped connectors)

Scope: The new `DELETE /connectors/{connector}/offsets` endpoint

Dependencies: The **STOPPED state** and **Resetting offsets (internal logic)** features

Test Plan

Unit, integration and/or system tests will be added for these cases:

Offsets tests:

- Cannot view the offsets of a nonexistent connector
- Can view the offsets of a running sink and source connector, and verify that that those offsets reflect an expected level of progress for each connector (i.e., they are greater than or equal to a certain value depending on how the connectors are configured and how long they have been running)
- Send the appropriate REST response when altering a connector's offsets depending on whether its `alterOffsets` method is not implemented, throws an `UnsupportedOperationException`, throws a different exception, or is implemented and returns successfully
- Cannot alter the offsets of a nonexistent, paused, or running connector
- Can alter the offsets of a stopped sink and source connector, verify that those offsets are returned in subsequent requests to view the offsets of the connector, and verify that those offsets are used by the connector when it is resumed
- Send the appropriate REST response when resetting a connector's offsets depending on whether its `alterOffsets` method is not implemented, throws an `UnsupportedOperationException`, throws a different exception, or is implemented and returns successfully
- Cannot reset the offsets of a nonexistent, paused, or running connector
- Can reset the offsets of a stopped sink and source connector
- Second follow-up requests to reset offsets for those connectors after their offsets have already been reset once are also successful
- Can view/alter/reset the offsets for a sink connector that uses an overridden consumer group ID
- Can view/alter/reset the offsets for a source connector that uses a custom offsets topic
- Can view/alter/reset the offsets for a sink and source connector that targets a different Kafka cluster than the one used for the internal topics of the Kafka Connect cluster

Stopped state tests:

- Can stop a running connector
- Can stop a paused connector
- Can stop a stopped connector (i.e., stopping an already-stopped connector is idempotent)
- Can delete a stopped connector
- Cannot see task configuration or status information in the REST API for a stopped connector
- Can resume a stopped connector
- Can pause a stopped connector
- Stopping a failed connector updates its state to `STOPPED` in the REST API
- Stopping a connector that fails during shutdown after receiving a stop request updates its state to `STOPPED` in the REST API
- Can resume a connector after its `Connector` has failed both before and during shutdown after receiving a stop request

Future work

Automatically delete offsets with connectors

Summary: Add a mechanism (such as an `includeOffsets` URL query parameter) to allow offsets to be reset at the same time a connector is deleted.

Delayed because: The cost of designing and implementing this feature is not high enough to justify its inclusion in the first draft of offsets support, though it may be a nice quality-of-life improvement later on. The difficulty here is finding a way to ensure that the connector and all of its tasks are stopped before its offsets are reset, since the existing endpoint to delete connectors does not synchronously wait for the connector to actually be deleted (only for the deletion request to be persisted in the config topic). One option might be to add some kind of offsets reset request record to the config topic directly after the tombstone for the connector config, but that brings its own difficulties, such as how to handle failures to reset offsets once the connector and its tasks have been shut down.

Rejected Alternatives

Source-only support

Summary: Only provide support for source connector offsets instead of both source and sink connector offsets, since there is existing CLI tooling for sink connector offsets provided already.

Rejected because: The additional cost of supporting sink connector offsets, at least with the APIs proposed here, is negligible, and it will be easier for Connect cluster administrators to interact with a single unified API for both source and sink connectors.

Full overwrite support

Summary: Either instead of or in addition to the `PATCH /connectors/{connector}/offsets` endpoint, add a `PUT /connectors/{connector}/offsets` endpoint that will cause the connector's offsets to fully match the offsets provided in the request body, by wiping any offsets not included in the request body if necessary.

Rejected because: This can be easily achieved with the existing APIs by first resetting the offsets for a connector, then altering its offsets.

Resetting offsets for active connectors

Summary: Allow offsets to be reset for currently-running connectors.

Rejected because: This would be significantly harder to implement, and the benefits are negligible compared to the workaround of stopping the connector, resetting its offsets, and then recreating it.