KIP-585: Filter and Conditional SMTs

- Status
- Motivation
- Public Interfaces
- Proposed Changes
    - Predicates
        - TopicNameMatches
        - HasHeaderKey
        - RecordIsTombstone
    - Conditionally applying an SMT
    - The Filter SMT
- Compatibility, Deprecation, and Migration Plan
- Rejected Alternatives

Status

Current state: Adopted

Discussion thread: *here*

Vote thread: *here*

JIRA: Unable to render Jira issues macro, execution error.

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

Motivation

Single Message Transformations (SMTs) in Kafka Connect provide a convenient, code-free way to modify records from source connectors before they get sent to a Kafka topic. In some more complex use cases it can be desired to apply SMTs dependent on some aspect of the record being processed.

For example, with Debezium there are topics which represent a schema change and topics which represent a data change, and users might want to apply transformations selectively, based on the topic type. SMTs cannot currently do this, since they're applied to all records produced by a source connector, irrespective of their intended topic. This problem would be solved if it was possible to apply an SMT according to the name of the topic (See KAFKA-7052 for further details).

This KIP proposes a way to only apply a particular transformation if the resource matches some condition. The condition is defined by a new interface and the implementations for common conditions will be provided. Connector authors and users will also be able to provide their own condition implementations for special cases, but this is not expected to be a common need.

A new `Filter` SMT will also be implemented. This will filter records which do, or do not, satisfy a given condition. This is advantageous for users who do not want to incur the storage costs of consuming everything from a source connector, for example. Instead they can chose to ingest only the records of interest. Likewise for sink connectors it will enable exporting a subset of data without needing to resort to a Kafka Streams application to filter it first.

Public Interfaces

A new `Predicate` interface will be added in the new `org.apache.kafka.connect.transforms.predicates` package.

```
/**
* A condition on ConnectRecords.
* Implementations of this interface can be used for filtering records and conditionally applying
Transformations.
* Implementations must be public and have a public constructor with no parameters.
*/
public interface Predicate<R extends ConnectRecord<R>> extends Configurable, AutoCloseable {

    /**
     * Configuration specification for this predicate.
     */
    ConfigDef config();

    /**
     * Returns whether the given record satisfies this predicate.
     */
    boolean test(R record);

    @Override
    void close();
}
```

All transformations will gain new implicit configuration parameters which will be consumed by the connect runtime and not passed to the `Transformation.configure()` method.

A new `Filter` SMT will be added to enable record filtering.

Proposed Changes

Predicates

The `Predicate` interface is described above. The interface will be a worker plug-in, loaded in the same way as other worker plug-ins such as converters, connectors, and REST extensions. This would include aliasing behaviour allowing users to specify predicates using their simple class names as long as no two predicate plug-ins with the same simple name are available on the worker.

In order to apply a transformation conditionally, all transformations will implicitly support a String `predicate` configuration parameter, which names a particular predicate.

To negate the result of a predicate, all transformations will implicitly support a boolean `negate` configuration parameter, which defaults to false.

In addition to the `Predicate` interface described above, this KIP will provide the following implementations:

TopicNameMatches

`test()` will return true when the `ConnectRecord.topic()` (i.e. it's name) matches a given Java regular expression pattern.

Config name	Type	Default	Required
pattern	String	null	yes

HasHeaderKey

`test()` will return true when the `ConnectRecord.headers()` has 1 or more headers with a given key.

Config name	Type	Default	Required
name	String	null	yes

RecordIsTombstone

`test()` will return true when the `ConnectRecord` represents a tombstone (i.e. has a null value). This predicate has no configuration parameters.

Conditionally applying an SMT

When a Transformation is configured with the new `predicate` parameter its application will happen conditionally. The value of the `predicate` parameter will be the name of a predicate defined under the `predicates` prefix. Configuration for the predicate will come from all other configuration parameters starting with the same `predicates...` analogous to how the transformations in a transformation chain are configured. These will be supplied to the `Predicate.configure(Map)` method.

If during processing the predicate throws an exception this will be handled in the same way as errors in transformations.

Consider the following example of a transformation chain with a single conditionally applied `ExtractField$Key` SMT:

```
transforms=t2
transforms.t2.predicate=has-my-prefix
transforms.t2.negate=true
transforms.t2.type=org.apache.kafka.connect.transforms.ExtractField$Key
transforms.t2.field=c1
predicates=has-my-prefix
predicates.has-my-prefix.type=org.apache.kafka.connect.predicates.TopicNameMatch
predicates.has-my-prefix.pattern=my-prefix-.*
```

The transform `t2` is only evaluated when the predicate `has-my-prefix` is false (the `negate` parameter). That predicate is configured by the keys with prefix `predicates.has-my-prefix.` The predicate class is `org.apache.kafka.connect.predicates.TopicNameMatch` and it's `pattern` parameter has the value `my-prefix-.*` . Thus the SMT will be applied only to records where the topic name does not start with `my-prefix-`.

The benefit of defining the predicate separately from the transform is it makes it easier to apply the same predicate to multiple transforms, or to have one set of transforms predicated on one predicate and another set of transforms predicated on that predicates negation.

The Filter SMT

A new `Filter` transformation will be added in the existing `org.apache.kafka.connect.transforms` package. This will return null from `apply(ConnectRecord)`. This is not of much use on its own, but is intended to applied conditionally as described above. This will allow messages to be filtered according to the predicate.

Consider the following example of a transformation chain with a single `Filter` SMT:

```
transforms=filter
transforms.filter.type=org.apache.kafka.connect.transforms.Filter
transforms.filter.predicate=foo-or-bar
predicates=foo-or-bar
predicates.foo-or-bar.type=org.apache.kafka.connect.transforms.predicates.TopicNameMatch
predicates.foo-or-bar.pattern=foo|bar
```

The predicate class is `org.apache.kafka.connect.predicates.TopicNameMatch` and it takes a single configuration parameter, `pattern`. Records having a topic name "foo" or "bar" match the predicate, so the `filter` SMT will be evaluated, will return null and therefore those records are filtered out.

Compatibility, Deprecation, and Migration Plan

Users will need to perform a rolling upgrade of a distributed connect cluster before they can start using the new `Filter` SMT or conditional SMTs.

Adding the new implicit `predicate` and `negate` parameters to transformations means that any existing transformation which already took config parameters of these names would not be configurable (i.e. the implicit parameters will mask the transformation parameters of the same name). Similarly existing connectors might have a configuration parameters prefixed by `predicates`, which would be masked by the new top-level parameter. The analogous situation arose when support for SMTs was originally added in KIP-66.

Rejected Alternatives

- Numerous alternative ways to configure conditional SMTs which reduced or removed the possibility of collision with existing connectors were considered. They were more verbose and difficult to understand.