KIP-802: Validation Support for Kafka Connect SMT and Converter Options

Status

Current state: *Under Discussion*

Discussion thread: *here*

JIRA: *KAFKA-13478*

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

Motivation

The existing Kafka Connect REST API method `PUT /connector-plugins/(string: name)/config/validate` lets users validate connector configuration. This is very useful to make sure that a connector configuration is valid before actually putting the connector in place.

No such validation functionality exists for the options of single message transforms (SMTs, KIP-66), SMT predicates (KIP-585), as well as (header) converters. I.e. the only way for a user to learn about invalid SMT- or converter-related configuration is to register a connector with that SMT or converter. This KIP aims at providing validation functionality for SMTs and converters, e.g. benefitting tools like UIs and CLIs interacting with Kafka Connect which can inform the user about invalid SMT or converter configuration early on.

Public Interfaces

A method `default Config validate(Map<String, String> smtConfigs)` will be added to the `org.apache.kafka.connect.transforms.Transformation` interface.

A method `default Config validate(Map<String, String> predicateConfigs)` will be added to the `org.apache.kafka.connect.transforms.predicates.Predicate` interface.

A method `default Config validate(Map<String, String> converterConfigs)` will be added to the `org.apache.kafka.connect.storage.Converter` interface.

A method `default Config validate(Map<String, String> headerConverterConfigs)` will be added to the `org.apache.kafka.connect.storage.HeaderConverter` interface.

The implementation of the existing REST API method `PUT /connector-plugins/(string: name)/config/validate` will be expanded so that it invokes the validate() method for all SMTs, predicates, and (header) converters specified in the given connector configuration and returns any validation errors as part of the existing response type structure.

Proposed Changes

The following method (inspired by the existing `org.apache.kafka.connect.connector.Connector.validate(Map<String, String>)` method) will be added to the interface `org.apache.kafka.connect.transforms.Transformation`:

```
/**
 * Validate the SMT configuration values against configuration definitions.
 * @param smtConfigs the provided configuration values
 * @return The updated configuration information given the current configuration values
 *
 * @since 3.2
 */
default Config validate(Map<String, String> smtConfigs) {

    ConfigDef configDef = config();

    if (null == configDef) {

        throw new ConnectException(

            String.format("%s.config() must return a ConfigDef that is not null.", this.getClass().getName())

        );

    }

    List<ConfigValue> configValues = configDef.validate(smtConfigs);

    return new Config(configValues);

}
```

The same method will be added to the `org.apache.kafka.connect.transforms.predicates.Predicate`, `org.apache.kafka.connect.storage.Converter`, and `org.apache.kafka.connect.storage.HeaderConverter` interfaces. Non-nullability for configDef must not be enforced in the header converter case as per this email discussion.

As this is a default method, this change is generally backwards compatible, i.e. existing `Transformation`, `Predicate`, `Converter`, and `HeaderConverter` implementations will continue to work as-is.

The method `org.apache.kafka.connect.runtime.AbstractHerder.validateConnectorConfig(Map<String, String>, boolean)` will be adjusted to:

- Determine the list of configured SMTs/predicates/converters from the connector configuration passed for validation
- Instantiate each configured SMTs/predicates/converters and invoke its `validate()` method with the subset of the connector configuration of that particular SMTs/predicates/converters
- Add the validation result(s) to the existing result type structure

This will make sure that the SMTs/predicates/converters attributes are validated upon explicit calls to the `validate` REST method as well as when a connector gets registered or updated.

Compatibility, Deprecation, and Migration Plan

Compatibility

The proposed change has generally good compatibility characteristics; the following compatibility aspects are pointed out:

Behavioral Compatibility

If a user had called the REST API's validation method with an invalid SMT configuration in the past, this would have been ignored, whereas in the future, this call will yield the corresponding validation errors. While this is a behavioral change, it is a desirable one, as users will benefit from the early validation feedback provided through this change. It seems highly unlikely that a user intentionally relied on SMT configuration failures to *not* be reported.

If an existing `Transformation`, `Predicate`, `Converter`, or `HeaderConverter` implementation declares a method with the exact same signature as the proposed `validate()` method, it will be invoked by `AbstractHerder.validateConnectorConfig()`, which may be surprising. Impact of this should be low, apart from the fact that the same validation routine will potentially be invoked multiple times.

Binary Compatibility

Apart from the following exception, the proposed change is binary compatible, i.e. an existing `Transformation`, `Predicate`, `Converter`, or `HeaderConverter` implementation class file compiled against an old version of the API can be used with the new API version without problems.

The exception is if the transformation implementation class "already implements another interface that declares a default method with a matching signature, and the client type already refers to the default method from the other interface (except when using the `Interface.super.method()` notation" (see here for details).

Chances for this to happen should be very low.

Source Compatibility

If an existing `Transformation, Predicate, Converter, or HeaderConverter` implementation declares

- a method `validate(Map<String, String> connectorConfigs)` with a return type other than `Config`, or
- a method `Config validate(Map<String, String> connectorConfigs)` with another visibility than `public`,

then a compilation error will be raised when compiling this implementation type against the new API version. The same is the case when recompiling the implementation class in the situation described above.

In these situations, the implementation type must be adjusted accordingly, e.g. by renaming that existing method.

Chances for this to happen should be very low.

Deprecation

As this is a new API, no deprecations are needed.

Migration

There are no migration concerns. In documentation, transformation authors may be encouraged to override the proposed new validation method with advanced custom validation logic if needed.

Rejected Alternatives

- *Alternative 1:* There could be a new, separate REST method, solely focused on validating SMT configuration. This seems disadvantageous, as a user then would have to make two REST API calls for validating an entire connector configuration
- *Alternative 2:* No API for validating SMT options gets added, keeping the current status quo; This seems disadvantageous, as users then continue to have no way for learning about invalid SMT configuration without actually registering a connector
- *Alternative 3:* Add a query parameter such as `?validate-only=true` to the existing connector registration endpoint; This seems disadvantageous, as it -- while providing the desired capability -- would be rather inconsistent with the current REST API design which defines separate endpoints for validation and registration