KIP-100 - Relax Type constraints in Kafka Streams API

- Status
- Motivation
- Public Interfaces
- Proposed Changes
- Compatibility, Deprecation, and Migration Plan
- Rejected Alternatives
- Notes

Status

Current state: Accepted

Discussion thread: here

JIRA: KAFKA-4481

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

Motivation

Several Kafka Streams methods currently take arguments that are functions parameterized in the key and value types to apply various transformations to KStreams and KTables. Those functions are currently invariant in their input and result types, when they should probably be contravariant in their key / value input types, and covariant in their result type.

For instance, `KStream<K, V>.filter(Predicate<K, V> predicate)` should be `KStream.filter(Predicate<? super K, ? super V> predicate)` to accept predicates that can act on any supertype of K, or V. More concretely, if `Cat` extends `Animal`, and I have `Predicate<Animal, Object> animalPredicate`, then I should be able to call `KStream<Cat, Picture>.filter(animalPredicate)`

Conversely for result types, `KStream<K, V>.map(ValueMapper<V, R> mapper)` should be `KStream<K, V>.map(ValueMapper<? super V, ? extends R> mapper)`. For example I can apply `ValueTransformer<Object, String> toStringTransformer` to `KStream<K, Serializable>.map(toStringTransformer)` and the result can safely be used as either `KStream<K, String>` or as `KStream<K, Serializable>` without relying on unchecked casts.

This change will make it easier to write reusable code for transformations, without requiring additional wrappers around existing code, or the unnecessary use of unchecked casts.

The same reasoning applies to the key, value and result types defined in methods that take `Aggregator`, `StreamPartitioner`, `KeyValueMapper`, `ValueMapper`, `ProcessorSupplier`, `TransformerSupplier`, `ValueTransformerSupplier`, `ForeachAction`, `StreamPartitioner`, and `ValueJoiner`.

Public Interfaces

Affected methods	Current argument type	New argument type
`(KGroupedStream|KGroupedTable).aggregate`	`Aggregator<K, V, T>`	`Aggregator<? super K, ? super V, T>`
`(KTable|KStream).filter*, KStream.branch`	`Predicate<K, V>`	`Predicate<? super K, ? super V>`
`(KStream|KTable).groupBy`	`KeyValueMapper<K, V, T>`	`KeyValueMapper<? super K, ? super V, T>`
`KStream.(selectKey|map|flatMap), KTable.toStream`	`KeyValueMapper<K, V, X>`	`KeyValueMapper<? super K, ? super V, ? extends X>`
`(KStream|KTable).mapValues, KStream.flatMapValues`	`ValueMapper<V, X>`	`ValueMapper<? super V, ? extends X>`
`KStream.transform`	`TransformerSupplier<K, V, X>`	`TransformerSupplier<? super K, ? super V, X>`
`KStream.transformValues`	`ValueTransformerSupplier<V, X>`	`ValueTransformerSupplier<? super V, X>`
`(KStream|Ktable).foreach`	`ForeachAction<K, V>`	`ForeachAction<? super K, ? super V>`
`KStream.process`	`ProcessorSupplier<K, V>`	`ProcessorSupplier<? super K, ? super V>`
`(KStream|KTable).*join`	`ValueJoiner<K, V, R>`	`ValueJoiner<? super K, ? super V, ? extends R>`

(KStream|KTable).(to|through)	StreamPartitioner<K, V>	StreamPartitioner<? super K, ? super V>
KafkaStreams.metadataForKey	StreamPartitioner<K, V>	StreamPartitioner<? super K, ? super V>

Proposed Changes

This KIP proposes changing the methods on the interfaces listed above to relax function arguments parameterized in key, value, and return types to accept super-types of those key and values, and sub-types of those return types.

For KGroupedStream/KGroupedTable `groupBy` and `aggregate` methods it was decided to leave the return type invariant, since the change is not as straightforward. Those methods sometimes require passing a `Serde<T>` or `Initializer<T>` where T needs to be consistent with the `Aggregator` (for aggregate) or `KeyValueMapper` (for groupBy) result type.

For backwards compatibility reasons, and to avoid runtime class cast exceptions, the choice was made to not make the result type covariant, even though that would have been more correct (see rejected alternatives).

Compatibility, Deprecation, and Migration Plan

- This change is binary compatible
- This change is source compatible for anyone merely calling the existing APIs
- This change is not source compatible for anyone extending the affected classes / interfaces.
- Update (2017-01-18): This change is not source compatible for anyone calling the Kafka Streams API from Scala due to differences in how Scala infers types.

Rejected Alternatives

For the aggregate and groupBy case the following alternatives would have been more correct – if we could drop support for Java 7 – since they would enforce the same type for initializer, serializer, and serde.

```
public <VR, VAGG extends VR> KTable<K, VR> aggregate(
    final Initializer<VAGG> initializer,
    final Aggregator<? super K, ? super V, VAGG> aggregator,
    final Serde<VAGG> aggValueSerde
);

<KR, KG extends KR> KGroupedStream<KR, V> groupBy(
    final KeyValueMapper<? super K, ? super V, KG> selector,
    final Serde<KG> keySerde,
    final Serde<V> valSerde

);
```

Unfortunately, when compiling against 1.7 source target, passing Aggregator<X, X, String> has the compiler incorrectly infer the result type as being KStream<T, Object>, whereas when compiling against 1.8 source target, the compiler correctly infers the result type as KStream<T, String>. It is still possible to coerce the 1.7 compiler into inferring the correct type by introducing an intermediate variable of type KStream<T, String>, or by explicitly casting to the correct type, however this makes it inconvenient to chain method calls. In addition, this would also break source compatibility for existing code compiled against 1.7 target.

In light of that we were forced to either:

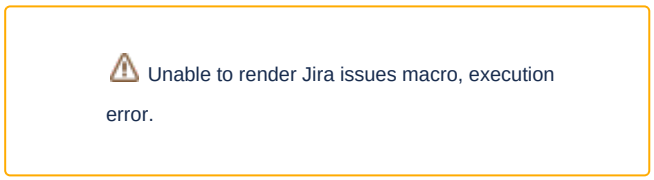
1. make no changes to the output type, i.e. keep the existing output type invariant, leaving the inconsistent API and do another API change once we can drop support for 1.7
2. use the more correct <T, VAGG extends T> contraint, and break source compatibility for 1.7 targets, forcing those users to rely on ugly casts or intermediate variables.
3. make the API consistent by making result types covariant using wildcards `? extends V`, relaxing compile time correctness across initializer, aggregator, and serde output types. We initially explored this route because if was backward compatible, but decided to drop it because it would a) require non-trivial changes to the existing streams code, and b) introduce lots of unchecked casts that could blow up at runtime if a user is not careful to ensure consistency across output types in aggregate / groupBy

We decided to chose approach 1. at the expense of a more consistent API, to ensure backwards compatibility for 1.7 users and avoid the complexity and potential pitfalls of the last approach.

Once we drop support for 1.7 we can always decide to switch to approach 2. without breaking source compatibility, by making a proposal similar to this KIP.

Notes

Update 2017-01-18: In light of



it was decided to leave return types invariant for `TransformerSupplier` and `ValueTransformerSupplier`