# KIP-832: Allow creating a producer/consumer using a producer/consumer config



## Status

The current KIP won't be implemented and will replaced by KIP-839 which will provide a better and elegant solution to the need.

**Current state**: *"Under Discussion"*

**Discussion thread**: https://lists.apache.org/thread/ooczzl9b1wo3gtm5z3vqokowkgo9rb48

**JIRA**: KAFKA-13864

## Motivation

To allow implementing Spring managed interceptors for producers and consumers, both, the *KafkaProducer* and the *KafkaConsumer* should expose constructors taking a config instance (*ProducerConfig* and *ConsumerConfig*) instead of a configuration given as a *Properties* or as a *Map*.

Having an access to the configuration object would give a way to override the existing method used to get interceptors from given class names.

This would also open the door for other uses cases currently using a class to get a concrete instance (partitioner, metrics report, etc).

A possible alternative would be to create more constructors with an additional parameter to specify the interceptors but this would increase the number of constructors without having a way to build instances for other types.

See the following portion of code for a concrete example

```
public class SpringAwareProducerConfig extends ProducerConfig {

    private final List<ProducerInterceptor<?, ?>> producerInterceptors;

    public SpringAwareProducerConfig(Map<?, ?> props,
                                     boolean doLog,
                                     List<ProducerInterceptor<?, ?>> producerInterceptors) {
        super(props, doLog);
        this.producerInterceptors = producerInterceptors;
    }

    @Override
    public <T> List<T> getConfiguredInstances(List<String> classNames, Class<T> t, Map<String, Object>
configOverrides) {
        final var configuredInstances = super.getConfiguredInstances(classNames, t, configOverrides);
        if (ProducerInterceptor.class.equals(t)) {
            configuredInstances.addAll((Collection<? extends T>) producerInterceptors);
        }
        return configuredInstances;
    }
}
```

```
public class SpringAwareConsumerConfig extends ConsumerConfig {

    private final List<ConsumerInterceptor<?, ?>> consumerInterceptors;

    public SpringAwareConsumerConfig(Map<?, ?> props,
                                     boolean doLog,
                                     List<ConsumerInterceptor<?, ?>> consumerInterceptors) {
        super(props, doLog);
        this.consumerInterceptors = consumerInterceptors;
    }

    @Override
    public <T> List<T> getConfiguredInstances(List<String> classNames, Class<T> t, Map<String, Object>
configOverrides) {
        final var configuredInstances = super.getConfiguredInstances(classNames, t, configOverrides);
        if (ConsumerInterceptor.class.equals(t)) {
            configuredInstances.addAll((Collection<? extends T>) consumerInterceptors);
        }
        return configuredInstances;
    }
}
```

Also see the Spring Kafka issue for the real motivation behind this KIP

https://github.com/spring-projects/spring-kafka/issues/2244

Kafka streams is not concerned by this issue as the KafkaStreams object is already exposing a constructor receiving a StreamsConfig object.

Concrete working example for Streams

```
public class SpringAwareStreamsConfig extends StreamsConfig {

    private final List<ProducerInterceptor<?, ?>> producerInterceptors;
    private final List<ConsumerInterceptor<?, ?>> consumerInterceptors;

    public SpringAwareStreamsConfig(Map<?, ?> props,
                                    boolean doLog,
                                    List<ProducerInterceptor<?, ?>> producerInterceptors,
                                    List<ConsumerInterceptor<?, ?>> consumerInterceptors) {
        super(props, doLog);
        this.producerInterceptors = producerInterceptors;
        this.consumerInterceptors = consumerInterceptors;
    }

    @Override
    public <T> List<T> getConfiguredInstances(List<String> classNames, Class<T> t, Map<String, Object>
configOverrides) {
        final var configuredInstances = super.getConfiguredInstances(classNames, t, configOverrides);
        if (ProducerInterceptor.class.equals(t)) {
            configuredInstances.addAll((Collection<? extends T>) producerInterceptors);
        }
        if (ConsumerInterceptor.class.equals(t)) {
            configuredInstances.addAll((Collection<? extends T>) consumerInterceptors);
        }
        return configuredInstances;
    }
}
```

Streams may also use another approach based on a custom KafkaClientSupplier and reusing this KIP

```
public class CustomKafkaClientSupplier extends DefaultKafkaClientSupplier {

    // additional dependencies given by constructor injection.

        @Override
        public Producer<byte[], byte[]> getProducer(final Map<String, Object> config) {
                return new KafkaProducer<>(<CUSTOM_CONFIG>, new ByteArraySerializer(), new
ByteArraySerializer());
        }

        @Override
        public Consumer<byte[], byte[]> getConsumer(final Map<String, Object> config) {
                return new KafkaConsumer<>(<CUSTOM_CONFIG>, new ByteArrayDeserializer(), new
ByteArrayDeserializer());
        }

        @Override
        public Consumer<byte[], byte[]> getRestoreConsumer(final Map<String, Object> config) {
                return new KafkaConsumer<>(<CUSTOM_CONFIG>, new ByteArrayDeserializer(), new
ByteArrayDeserializer());
        }

        @Override
        public Consumer<byte[], byte[]> getGlobalConsumer(final Map<String, Object> config) {
                return new KafkaConsumer<>(<CUSTOM_CONFIG>, new ByteArrayDeserializer(), new
ByteArrayDeserializer());
        }
}
...
new KafkaStreams(<TOPOLOGY>, <PROPERTIES>, <CUSTOM_KAFKA_CLIENT_SUPPLIER>);
...
```

See another similar KIP for additional information related to the current KIP but for streams.

KIP-378: Enable Dependency Injection for Kafka Streams handlers

Having a new constructor and increasing the visibility of an existing constructor is quite straight and does not impact the existing. It gives an additional way to build a producer and a consumer. This approach is also align with Streams already having such kind of constructor.

Important note: this request is not related to Spring and would give a way to use constructed instances instead of using reflection to build these instances from a given class.

# Proposed Changes

Create 2 new constructors in *KafkaProducer*

```
public KafkaProducer(ProducerConfig config)
public KafkaProducer(ProducerConfig config, Serializer<K> keySerializer, Serializer<V> valueSerializer)
```

and create a new constructor + increase the visibility of an existing constructor in *KafkaConsumer*

```
public KafkaConsumer(ConsumerConfig config)
public KafkaConsumer(ConsumerConfig config, Deserializer<K> keyDeserializer, Deserializer<V> valueDeserializer)
```

The 2 following constructors will also be added into the *TopologyTestDriver*

```
public TopologyTestDriver(Topology topology, StreamsConfig config)
public TopologyTestDriver(Topology topology, StreamsConfig config, Instant initialWallClockTime)
```

# Compatibility, Deprecation, and Migration Plan

No compatibility issue and no migration plan are required as this KIP will only create new constructors and will increase the visibility of an existing constructor.

So, at the end, users will have more ways to build producers, consumers and a topology test driver without any impacts on the existing.

# Rejected Alternatives

/