KIP-873: ExceptionHandlingDeserializer, RetryDeserializer, PipeSerializer, PipeDeserializer

- Status
- Motivation
- Public Interfaces
- Proposed Changes
    - ExceptionHandlingDeserializer
        - RetryDeserializer
        - PipeDeserializer/PipeSerializer
- Compatibility, Deprecation, and Migration Plan
- Test Plan
- Rejected Alternatives

Status

Current state: *Under Discussion*

Discussion thread: None

JIRA: None

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

Motivation

At Intuit, we read messages of Kafka and decrypt using an in-house deserializer. This deserializer can throw an exception, either temporary (due to loss of network connection) or permanent (e.g. corrupted data), in which case the KafkaConsumer throws the exception.

Recovery is not straight-forward.

In the permanent case, we just want to skip the message (maybe writing it to a dead-letter queue).

In the temporary case, we might wish to retry the decryption before either giving up, or re-throwing exception.

We want to combine that with support for deserialization to CloudEvents.

It would be good to establish a pattern for this, rather than each team or organization coming up with their own solutions.

- The community benefits from having gotten it right, once, one time.
- Everyone understands how they work.
- Well thought out by the Kafka team.
- No proprietary solutions.

Public Interfaces

- org/apache/kafka/common/serialization/ExceptionHandlingDeserializer
- org/apache/kafka/common/serialization/RetryDeserializer
- org/apache/kafka/common/serialization/PipeDeserializer
- org/apache/kafka/common/serialization/PipeSerializer

Proposed Changes

I propose to introduce three new deserializers and one new serializer.

They all follow the decorator pattern.

ExceptionHandlingDeserializer

This class wraps a delegate deserializer specified by a property:

```
exception.handling.deserializer.delegate=org.apache.kafka.common.deserialization.ListDeserializer
```

To deserialize messages, it calls the delegate. If the delegate throws an exception, it return an object contaiing the exception, if there is no exception, then it simple returns the result.

We only want to support some exception (the permanent ones), so we need a matching strategy.

```
exception.handling.deserializer.matcher=org.apache.kafka.common.deserialization.ExceptionHandlingDeserializer.
MatchAllExceptions
```

The matcher needs to implement a functional interface:

```
@FunctionalInterface
public interface Matcher {
  boolean matches(Exception e);
}
```

Proof of concept PR

RetryDeserializer

Again, this wraps a delegate deserializer specified by a property:

```
retry.deserializer.delegate=org.apache.kafka.common.deserialization.ListDeserializer
retry.deserializer.backoff=org.apache.kafka.common.deserialization.RetryDeserializer.ExponentialBackoff
# https://docs.aws.amazon.com/step-functions/latest/dg/concepts-error-handling.html#error-handling-retrying-after-
an-error
retry.deserializer.backoff.exponential.intervalSeconds=1
retry.deserializer.backoff.exponential.maxAttempts=5
retry.deserializer.backoff.exponential.backoffRate=1.2
```

The backoff policy is configurable and pluggable. One strategy would be no back-off attempts. No back-off would be suitable for exceptions that are permanent.

PipeDeserializer/PipeSerializer

The above deserializer need to be composed together to create more complex serialization and deserialization. For example,

1. Decrypt message into bytes
2. Wrap that decryption in retry.
3. Wrap the retry in exception handling.
4. Convert the result into an object (e.g. a CloudEvent using CloudEventsDeserializer).

Putting that all together:

```
key.deseriailzer=org.apache.kafka.common.deserialization.ExceptionHandlingDeserializer
exception.handling.deserializer.delegate=org.apache.kafka.common.deserialization.RetryDeserializer
retry.deseriailzer.delegate=org.apache.kafka.common.deserialization.PipeDeserializer
pipe.deserializer.from=com.example.DecryptionDeserializer
pipe.deserializer.to=io.cloudevents.kafka.CloudEventsDeserializer
```

Proof of concept PR

Compatibility, Deprecation, and Migration Plan

- *New deserializers that support many common patterns.*
- *Well understood*

Test Plan

These can all be unit tested. They also need to be battle-tested with users.

Rejected Alternatives

Anyone can build themself in-house monolithic deserializer, but that's inflexible and results in deserializer explosion, ultimately you end up with CloudEvents DecryptingRetryExceptionHandlingDeserializier. I hope I don't need to explaining. LMK if you don't understand why that is undesirable.