KIP-917: Additional custom metadata for remote log segment

Status

Current state: *Accepted*

Discussion thread: *here*

JIRA: *here*

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

Motivation

KIP-405: Kafka Tiered Storage introduced the `RemoteLogSegmentMetadata` class which describes a log segment uploaded to the remote storage. It contains useful bits of metadata such as the segment ID, start and end offsets, etc. It is (will be) created by the broker side and implementations of `RemoteStorageManager`--i.e. remote storage plugins--don’t have the possibility to add any custom bit of metadata to be stored and propagated along with the standard ones. However, this might be useful in certain circumstances, for example:

1. A `RemoteStorageManager` makes the decision how and where to store the segment. Imagine a situation where it needs to apply a certain load balancing strategy across buckets on AWS S3 or Google Cloud Storage or storage accounts on Azure Blob Storage. If the strategy is deterministic, its inputs (e.g. the number of buckets to balance across) may change with time. It can also be non-deterministic, i.e. randomized. In this case, storing the name of the bucket or another identifier along with other remote log segment metadata is essential for the segment to be retrievable later.
2. In some cases, it’s necessary to know how much of the remote storage is consumed by a topic or a partition. `RemoteLogSegmentMetadata` has the `segmentSizeInBytes` field. However, this field includes only the segment itself without indices or any other files that may be uploaded together with the segment. Besides, `segmentSizeInBytes` represents the size of the raw file size on the local disk, which doesn’t take into account potential transformations (e.g. compression, encryption) a `RemoteStorageManager` may apply to the data.

Considering that remote segments are already pretty coupled with the remote storage plugin that wrote them, adding the custom metadata won't introduce new coupling.

Proposed Changes

This KIP proposes to add the possibility for `RemoteStorageManager.copyLogSegmentData` method to return optional custom metadata after a segment has been uploaded. This piece will be stored and propagated along with the standard metadata. The custom metadata will be a opaque byte array to the rest of the system. It should be serialized, deserialized, and interpreted by the remote storage plugin only. The `kafka-clients` library, which is distributed separately from Kafka and can be used as a plugin dependency, has convenient tools for binary serde.

Allowing remote storage plugins to write custom metadata of arbitrary size is not good for the stability of the cluster. The broker should have a configurable limit for their size. If a piece of custom metadata exceeds the limit, the execution of `RLMTask.copyLogSegmentsToRemote` should be interrupted with an error message. The default value for the limit should be 128 B, which is small enough to not be disruptive.

It's proposed to return the custom metadata from `RemoteStorageManager.copyLogSegmentData`. Hence, the custom metadata size in fact will be checked after the segment data is copied to the remote storage. In case the size limit is exceeded, one best effort attempt will be made to delete the data from the remote storage. Considering the situation with exceeding the limit should be very rare and abnormal and will require an operator actions, it's proposed to not introduce a complex cleanup mechanism or change the interface of `RemoteStorageManager` to e.g. precompute custom metadata and return to the broker side for inspection. (Put this also into *Rejected Alternatives*.)

See *Public Interfaces* for the details.

Public Interfaces

`RemoteLogSegmentMetadata.CustomMetadata`

Introduce a class for storing custom metadata.

RemoteLogSegmentMetadata.CustomMetadata

```
public class RemoteLogSegmentMetadata extends RemoteLogMetadata {

    public static class CustomMetadata {
        private final byte[] value;

        public CustomMetadata(byte[] value) {
            this.value = value;
        }

        public byte[] value() {
            return value;
        }
    }

...
```

`RemoteStorageManager.copyLogSegmentData`

This method will return `Optional<CustomMetadata>` instead of `void`. The documentation will be adjusted accordingly.

RemoteStorageManager.copyLogSegmentData

```
/**
 * ...
 * @return Custom metadata to be added to the segment metadata after copying.
 * ...
 */
Optional<CustomMetadata> copyLogSegmentData(
  RemoteLogSegmentMetadata remoteLogSegmentMetadata,
  LogSegmentData logSegmentData
) throws RemoteStorageException;
```

`RemoteLogSegmentMetadata` class

This class will have a new method:

customMetadata

```
/**
* @return Optional map of custom metadata for this segment.
*/
public Optional<CustomMetadata> customMetadata() {
   return customMetadata;
}
```

The corresponding private field will be created, the constructors and the documentation will be adjusted accordingly.

There should be a possibility to create an instance with this field being `Optional.empty()` (primarily, for the initial passing to `RemoteStorageManager.copyLogSegmentData`).

The method `createWithCustomMetadata` will be added for creating a new instance from an existing one with the provided custom metadata:

createWithCustomMetadata

```
public RemoteLogSegmentMetadata createWithCustomMetadata(CustomMetadata customMetadata) {
   return new RemoteLogSegmentMetadata(remoteLogSegmentId, ..., customMetadata);
}
```

`RemoteLogSegmentMetadataSnapshot` class

The same field and constructor changes as in the `RemoteLogSegmentMetadata` class.

RemoteLogSegmentMetadataRecord

This record definition will have a new field:

RemoteLogSegmentMetadataRecord

```
{
  "name": "CustomMetadata",
  "type": "bytes",
  "default": "null",
  "versions": "0+",
  "nullableVersions": "0+",
  "about": "Custom metadata."
}
```

`RemoteLogSegmentMetadataUpdateRecord`

Same as for `RemoteLogSegmentMetadataRecord`.

`RemoteLogSegmentMetadataSnapshotRecord`

Same as for `RemoteLogSegmentMetadataRecord`.

Configuration Keys

Key Name	Description	Valid Values	Default Value
`remote.log.metadata.custom.metadata.max.bytes`	The maximum size of custom metadata in bytes that the broker should accept from a remote storage plugin. If custom metadata exceeds this limit, the updated segment metadata will not be stored, the copied data will be attempted to delete, and the remote copying task for this topic-partition will stop with an error.	`0..Integer.MAX_VALUE`	128

Compatibility, Deprecation, and Migration Plan

Since the tiered storage functionality is not implemented yet and the related interfaces are evolving, it’s proposed to not preserve backward compatibility.

No special migration process or tool is needed.

Test Plan

The changes will be tested on the unit level, the existing unit tests will be adjusted.

Rejected Alternatives

1. Introduce a separate storage for custom metadata. This will not reduce the coupling or be better in any other visible way, but will make the solution more complex (e.g. will require a separate cache and a correlation mechanism to match custom and regular metadata) and more difficult to operate.

2. Change the interface of `RemoteStorageManager` to introduce a separate method for calculating the custom segment metadata before the attempt to copy it to the remote storage. This would allow to do the check the custom metadata size before an attempt to copy the segment. However, the situation with exceeding the custom metadata size limit is abnormal, should be very rare, and requires an operator intervention anyway, it's decided to not make the solution unnecessarily complex.