KIP-950: Tiered Storage Disablement

This page is meant as a template for writing a KIP. To create a KIP choose Tools->Copy on this page and modify with your content and replace the heading with the next KIP number and a description of your issue. Replace anything in italics with your own description.

Authors: Mehari Beyene, Divij Vaidya

Collaborators: Karthik Rajagopalan, Nagarjuna Koduru

Status

Current state:*Under Discussion*

Discussion thread: *here*

JIRA: *KAFKA-7739*

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

Motivation

This proposal aims to address a gap in the existing functionality introduced by KIP 405, which enabled customers to enable Tiered Storage but did not provide the capability to disable it. To enhance the flexibility and control of data storage in Kafka clusters, we propose the introduction of a Tiered Storage disablement feature. This feature will allow customers to selectively disable Tiered Storage on a per-topic basis. There are several reasons why customers would be interested in this feature:

1. **Flexibility**: Currently, customers can enable Tiered Storage but lack the ability to disable it, creating a one-way decision that may discourage them from fully exploring the feature. By introducing the Tiered Storage disablement feature, we offer customers the flexibility to enable or disable the feature as needed, empowering them to make informed choices that align with their specific requirements.
2. **Cost Optimization**: When data is duplicated in both local and tiered storage, it incurs costs for both tiers. However, not all customers require long-term tiered storage for their data. By providing the option to disable the feature, customers can save costs when Tiered Storage is not necessary, allowing them to optimize their expenses without compromising data retention needs.
3. **Temporary or Experimental Setups**: The Tiered Storage disablement feature grants customers the ability to conduct temporary experiments and evaluate the benefits of tiered storage without committing to a permanent architectural change in their data pipelines. By enabling customers to disable Tiered Storage when not needed, they can easily evaluate the impact and advantages of the feature, fostering a culture of experimentation and informed decision-making.
4. **Compatibility**: Allowing customers to disable the Tiered Storage feature enhances the compatibility of Kafka clusters. It enables seamless migration between different Kafka versions or service providers that may not support Tiered Storage. By offering the option to disable Tiered Storage, customers can smoothly transition their Kafka deployments, ensuring compatibility and reducing potential operational challenges.

Goals/Requirements

1. **Disable Tiering on a Topic**: Customers should have the ability to disable tiering on a topic that has Tiered Storage enabled.

2. **Retention of Tiered Data**: When disabling tiered storage on a topic, customers should have the option to retain the tiered data according to the configured retention policy prior to disabling tiering.
3. **Deletion of Tiered Data:** When disabling tiered storage on a topic, customers should have the option to delete the tiered data in the remote storage associated with that topic.
4. **Fetching Previously Tiered Data**: Customers should have the option to continue fetching data that was previously tiered before the tiered storage disablement. This allows them to access and consume the tiered data even after the tiering feature has been disabled.
5. **Re-enabling Tiering on a Previously Disabled Topic**: Customers should be able to re-enable tiering on a topic that was previously disabled, allowing them to resume tiered storage for that specific topic.
6. **Support for Both Kraft and ZooKeeper-Based Clusters**: The tiered storage disablement feature should be designed and implemented to support both Kraft and ZooKeeper-based Kafka clusters, ensuring compatibility across different cluster configurations.

Non-Goals

The following aspects are considered as non-goals for the tiered storage disablement feature:

1. **System-Wide Disablement**: The tiered storage disablement feature does not aim to support the system-wide disablement of tiered storage on a cluster level, as this is controlled by the cluster-level configuration `remote.log.storage.system.enable`.
2. **Support for Compacted Topics**: Tiered Storage is not supported for compacted topics, and this behavior will remain unchanged. During re-enablement, the feature will not support tiered storage for compacted topics. When initially enabling tiered storage for a topic, we check that the topic does not have compaction enabled, however, if the topic has historically used a compaction policy, we do not perform deep check to restrict tiered storage enablement. This behavior will remain consistent during re-enablement as well.

Proposed Changes

Concepts

Remote Log Disablement Policy

When disabling tiered storage on a topic, users will have the option of retaining or deleting the remote log at the time of disablement. This is represented by a remote log disablement policy. It will be an optional policy that users can specify. By default, we will retain the remote log.

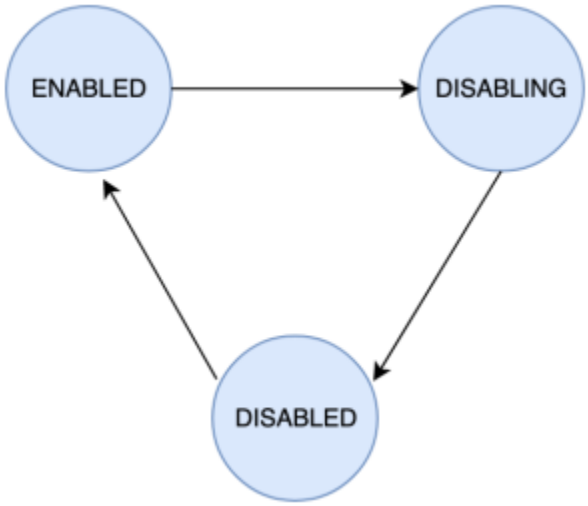
- *remote.log.disable.policy=retain*: Logs that are archived in the remote storage will be part of the contiguous "active" log. Clients can fetch the archived log, but no new data will be archived to the remote storage.
- *remote.log.disable.policy=delete:* Logs that are archived in the remote storage will not be part of the contiguous "active" log and will be deleted asynchronously as part of the disablement process.

Tiered Epoch

The tiered epoch signifies the version of a tiered storage topic. This epoch establishes a boundary for the transition from the enabling to disabling state, preventing concurrent and out-of-order modifications. Its aim is to guarantee that all activities within the disablement process are fully completed before permitting the re-enablement of tiering.

State Transitions

When users disable tiered storage on a topic, the tiered storage state on a topic transitions through the following stages: ENABLED DISABLING DISABLED.



ENABLED
The `ENABLED` state represents when tiered storage is enabled on a topic or when it has been re-enabled after disablement. In this state, the RemoteLogManager (RLM) has scheduled a task for the topic-partitions with the RemoteLogManager thread pool and RemoteStorageFetcher thread pool. The possible state transition from the `ENABLED` state is to the `DISABLING` state.

DISABLING
When users initiate an alter config API call to disable tiered storage on a topic that is currently in the `ENABLED` state, the topic enters the `DISABLING` state . In this state, various tasks are performed asynchronously to complete the disablement process. These tasks include:

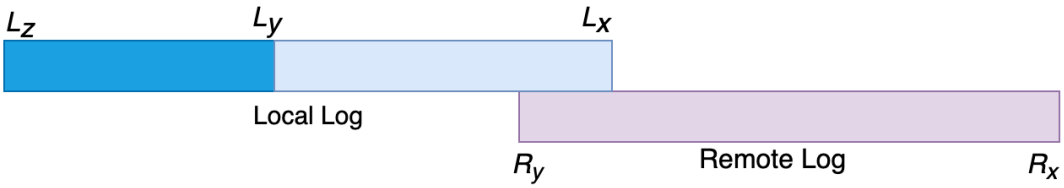
- Cancel tasks for the topic-partitions from the RemoteLogManager thread pool to stop archiving logs to the remote storage.
- If the disablement was triggered with the `remote.log.disable.policy=delete` option, draining fetch requests in RemoteFetchPurgatory and block new fetch requests from getting added to the RemoteStorageFetcher thread pool.
- If the disablement was triggered with the `remote.log.disable.policy=retain` option, we will cancel the tasks for the topic-partitions from the RemoteLogManager thread pool only for stopping archiving logs but the task for expiring remote segments will continue to work. The fetch requests from remote storage will also continue to work after disablement

While the topic is in the `DISABLING` state, new alter config API calls to disable tiered storage will be rejected. The possible state transition from the `DISABLING` state is to the `DISABLED` state.

DISABLED
The `DISABLED` state represents when tiered storage is fully disabled on a topic. In this state, the RLM does not have any scheduled tasks for the topic-partitions in the RemoteLogManager thread pool for copying segments to remote storage. If the topic was disabled with the `remote.log.disable.policy=delete` option, all remote data is marked for deletion and it is inaccessible for read. If the topic was disabled with the `remote.log.disable.policy=retain` option, there will be tasks scheduled for the topic-partitions only for expiration of remote logs but not for copying logs to remote storage. The RemoteStorageFetcher thread pool will also continue to accept remote fetch requests. The possible state transition from `DISABLED` state is to the `ENABLED`.

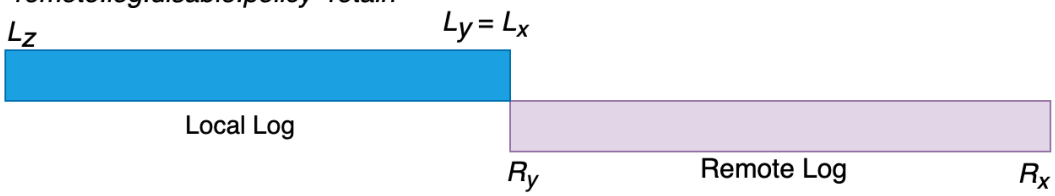
Data Expiry

Local and remote log offset before disablement

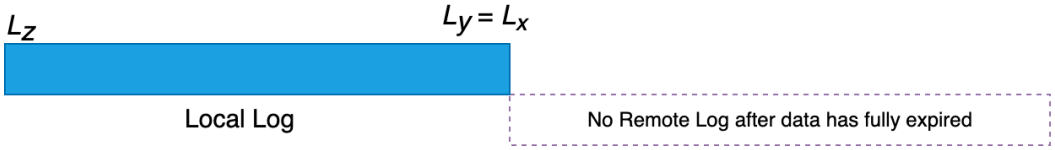


Local and remote log offset after disablement

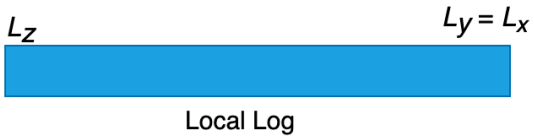
remote.log.disable.policy=retain



after disablement time + retention time all remote data will expire



remote.log.disable.policy=delete



Lx = *Local log start offset* L_z = *Local log end offset* L_y = *Last stable offset(LSO)*

R_y = *Remote log end offset* R_x = *Remote log start offset*

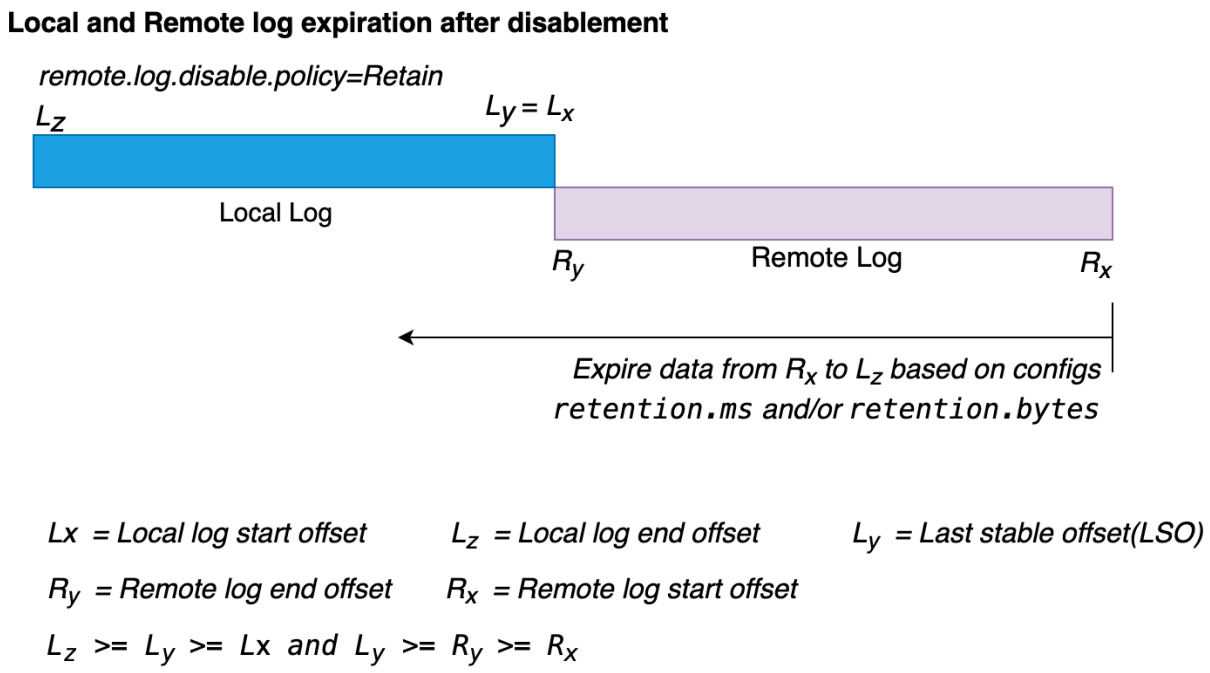
$L_z >= L_y >= Lx$ *and* $L_y >= R_y >= R_x$

When tiered storage is enabled in a topic, there are two sets of retention configurations available: **local retention configuration** (`local.retention.ms` and/or `local.retention.bytes`) and **topic-wide retention configuration**(`retention.ms` and/or `retention.bytes`). For the topic-wide retention configuration, if the values are not explicitly set, Kafka falls back to using the `log.retention.ms` and/or `log.retention.bytes` values.

When tiered storage is disabled on a topic, the local retention configuration becomes irrelevant, and all data expiration follows the topic-wide retention configuration exclusively.

If tiered storage is disabled using the `remote.log.disable.policy=delete` policy, the topic will not have any archived segments in remote storage. In this case, the topic-wide retention configuration operates similarly to any non-tiered storage topic.

On the other hand, if tiered storage is disabled with the `remote.log.disable.policy=retain` policy, it is possible that the topic still has segments in remote storage. In such scenarios, the topic-wide retention configuration will come into effect starting from the Remote log start offset (R_x).



As depicted in the diagram above, when tiered storage is disabled for a topic and there are still segments in the remote storage, the data expiration takes place from right (R_x) to left (L_z) based on the topic-wide retention settings of `retention.ms` and/or `retention.bytes`. This expiration process is relevant only when tiered storage is disabled with the `remote.log.disable.policy=retain`, and there are remaining segments in the remote storage. If tiered storage is disabled with `remote.log.disable.policy=delete` or if there is no data in the remote storage, the expiration behaves the same as for non-tiered storage topics.

Re-enabling Tiered Storage

This KIP supports the ability to re-enable tiered storage after it has been disabled. When tiered storage is enabled and disabled multiple times for a topic, the log offset remains continuous without any gaps. This is achieved by ensuring that data expiration always starts from the earliest offset, regardless of the storage tier being used.

To ensure offset continuity, when tiered storage is disabled with a `remote.log.disable.policy=retain` policy, logs are expired starting from the remote log start offset, and only after all segments from remote storage have expired, local logs are expired. By following this approach, the continuity of offsets is preserved during the re-enablement.

Considering different scenarios, we can examine the continuity of log offsets:

Scenario 1: Re-enabling after tiered storage is disabled with `remote.log.disable.policy=delete`. In the majority of cases, this is equivalent to enabling tiered storage for the first time as there will be no remote data from previous tiered storage enablement. However, it is possible that the asynchronous deletion of remote data may still be in progress when tiered storage is re-enabled from the previous disablement. To ensure there is no data mixup between the ongoing deletion and the newly tiered data after re-enablement, we take two measures. First, during disablement, we mark the segments for deletion and update the log start offset, and second, we fence the segments with the tiered epoch. Depending on the local retention configuration, data will be tiered to remote storage. Hence, even if the segments marked for deletion from the previous tiered storage enablement are still present in the remote storage awaiting deletion, they are completely isolated from the new data that is being tiered.

Scenario 2: Re-enabling after tiered storage is disabled with `remote.log.disable.policy=retain` and there is no data in remote storage because all previously tiered data has expired. This scenario is similar to scenario #1, as all the data in remote storage has already expired. Following the local retention configuration, data will be tiered to remote storage.

Scenario 3: Re-enabling after tiered storage is disabled with `remote.log.disable.policy=retain` and there is data in remote storage. When tiered storage was disabled, the remote log end offset (R_y) = local log start offset (L_x) + 1. After re-enabling tiered storage, the offset of the newly archived segments will continue from Lx, based on the local retention settings. This continuity is ensured by expiring data starting from the remote logs start offset (R_y), even while tiered storage is disabled. Local logs are only expired if there is no data in remote storage, guaranteeing offset continuity during the re-enabling process.

Enablement Code Path - Existing KIP-405 Flow

When the broker starts up, it checks the value of the configuration parameter `remote.log.storage.system.enable`. If it is enabled, the broker initializes a RemoteLogManager (RLM). The RLM then initializes two pluggable components: RemoteStorageManager (RSM) and RemoteLogMetadataManager (RLMM). It also sets up two thread pools: `RemoteLogManagerScheduledThreadPool`, responsible for copying log segments to remote storage and expiring remote segments, and `RemoteStorageFetcherThreadPool`, responsible for handling fetch requests that require offsets in remote storage.

Each topic can be configured to use remote storage by setting the configuration `remote.storage.enable` for that particular topic. This configuration can be set during topic creation or modified after the topic has been created.

If the cluster is using ZooKeeper (Zk) as the control plane, enabling remote storage for a topic triggers the controller to send this information to ZooKeeper. Each broker listens for changes in Zk, and when a change is detected, the broker triggers RemoteLogManager#onLeadershipChange().

In Kraft mode, when remote storage is enabled for a topic, the controller sends the information to the broker as a topic metadata change record. Each broker applies the metadata delta locally and calls RemoteLogManager#onLeadershipChange().

When RemoteLogManager#onLeadershipChange() is called, it schedules a leader or follower task in the RemoteLogManagerScheduledThreadPool for each topic partition that has tiered storage enabled.

Disablement code Path

The disablement code path differs between ZooKeeper-based clusters and Kraft-based clusters. However, the interface for communicating with the RemoteLogManager and how the RemoteLogManager handles disablements remain the same for both types of clusters. In this section, we will discuss the changes in the RemoteLogManager and the disablement code path for both ZooKeeper-based and Kraft-based clusters.

RemoteLogManager Change

The RemoteLogManager has two separate thread pools: RemoteLogManagerScheduledThreadPool (for copying and expiring remote segments) and RemoteStorageFetcherThreadPool (for fetching remote data).

When tiered storage is disabled with `remote.log.disable.policy=delete`, there is no remote data to expire or read from. Therefore, all scheduled tasks for the topic-partition from the RemoteLogManagerScheduledThreadPool can be canceled. However, when tiered storage is disabled with `remote.log.disable.policy=retain`, even though segments are not copied to remote storage, tasks are still needed for expiring remote data and serving fetch requests for the remote data. Thus, even when tiered storage is disabled on a specific topic, there will still be tasks for expiring remote data and fetching data from the remote storage.

To seamlessly support this functionality, we propose splitting the RemoteLogManagerScheduledThreadPool in RemoteLogManager into two separate thread pools: **RemoteStorageCopierThreadPool** and **RemoteDataExpirationThreadPool**.

The responsibilities of these thread pools will be as follows:

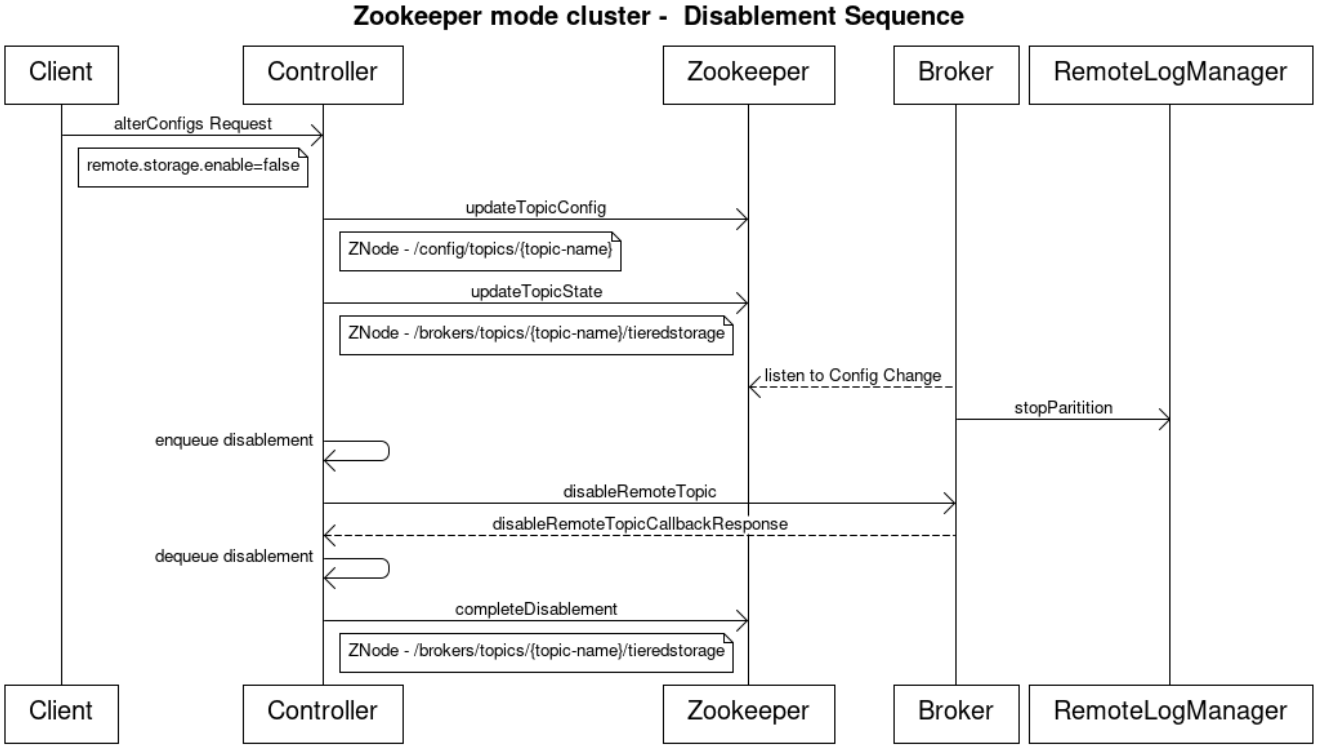
- RemoteStorageCopierThreadPool: This thread pool will be responsible for topic-partition tasks related to copying segments to remote storage.
- RemoteDataExpirationThreadPool: This thread pool will be responsible for topic-partition tasks related to expiring remote segments.

During tiered storage enablement, when RemoteLogManager#onLeadershipChange() is called, tasks for the topic-partitions will be scheduled in these thread pools, similar to how we do it today in the RemoteStorageFetcherThreadPool.

During tiered storage disablement, when RemoteLogManager#stopPartition() is called:

- Tasks scheduled for the topic-partitions in the RemoteStorageCopierThreadPool will be canceled.
- If the disablement policy is `delete`, tasks scheduled for the topic-partitions in the RemoteDataExpirationThreadPool will also be canceled.
- However, if the disablement policy is `retain`, scheduled tasks for the topic-partitions in the RemoteDataExpirationThreadPool will remain unchanged.

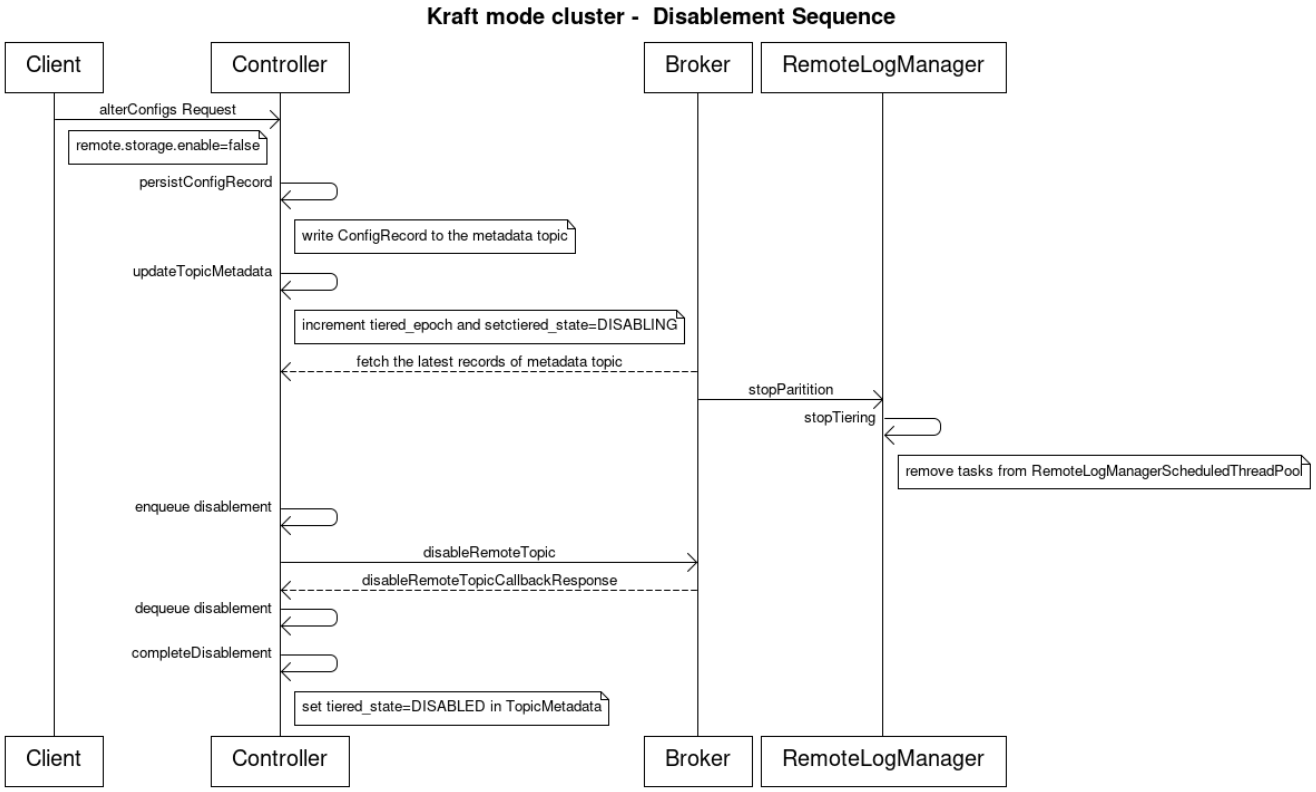
Disablement - Zookeeper Based Cluster



To disable tiered storage on a topic in ZooKeeper-based clusters, the following steps are involved:

1. Users modify the topic configuration:
    a. Users set the configuration `remote.storage.enable=false` for the desired topic, indicating the disablement of tiered storage.
    b. Optionally, users specify a disablement policy to either retain or delete the remote data associated with the topic.remote.log.disable.policy=retain | delete. The disablement policy defaults to Retain if the option is not specified.
2. Controller triggers configuration change in ZooKeeper:
    a. Disabling tiered storage on a topic prompts the controller to send the configuration change to ZooKeeper.
    b. The topic's configuration in the config ZNode is updated to `remote.storage.enable=false` .
    c. The tiered_epoch is incremented, and the tiered_state is set to DISABLING under the topics ZNode.
3. Broker detects the configuration change:
    a. Each broker in the cluster listens to changes in ZooKeeper.
    b. Upon detecting that the tiered storage configuration for a topic is disabled, the broker invokes the RemoteLogManager#stopPartitions() method.
4. Execution of RemoteLogManager#stopPartitions():
    a. Removes the scheduled tasks for the topic-partition from the thread pool responsible for archiving logs to remote storage.
    b. If the disablement policy is set to `delete`, the Log start offset (LSO) is updated to match the Local Log Start Offset and the remote log is deleted by calling the `RemoteStorageManager#deleteLogSegmentData()`.
    c. If the disablement policy is set to `retain`, there is no extra call required to delete remote data as the RemoteLogManager leader partition task will periodically expire segments that are earlier than the log start offset.
5. Controller interaction:
    a. The controller enqueues the disablement intent for tiered storage to a local queue.
    b. It sends an API call, called DisableRemoteTopic, to each broker with a callback to disable the remote topic.
    c. DisableRemoteTopic is a new controller to broker API that will be introduced in this KIP
    d. Note that the current approach for communication between the Controller and brokers involves a familiar pattern. Tasks are enqueued in a local queue within the Controller Context, and a request is sent to the brokers along with a callback mechanism. This method is currently employed by the controller for operations, such as deleting topics. For instance, when the controller initiates the deletion of topics, it maintains a local queue called `topicsToBeDeleted`. This queue serves the purpose of monitoring the progress of topic deletions.
6. Completion of disablement:
    a. Once the controller receives responses from all brokers for the DisableRemoteTopic request, it updates the tiered_state in the topics ZNode.
    b. This update marks the completion of the disablement process, indicating that tiered storage has been successfully disabled for the topic in ZooKeeper-based clusters.

Disablement - Kraft Based Cluster



To disable tiered storage on a topic in Kraft-based clusters, the following steps are involved:

1. Users modify the topic configuration:
    a. Users set the configuration "remote.storage.enable=false" for the desired topic, indicating the disablement of tiered storage.
    b. Optionally, users specify a disablement policy to either retain or delete the remote data associated with the topic: `remote.log.disable.policy=retain|delete`.
2. Controller persists configuration change:
    a. The controller creates a ConfigRecord and persists it in the metadata topic.
    b. The controller also updates the Topic metadata to increment the `tiered_epoch` and update the `tiered_state`to `DISABLING` state.
3. Broker detects the configuration change:
    a. Broker replicate the metadata topic and fetch the latest records of ConfigRecord changes.
    b. The broker calls the ConfigHandler to process the ConfigRecord change with tiered storage disablement.
    c. The broker ConfigHandler calls `RemoteLogManager#stopPartitions()`.
4. Execution of `RemoteLogManager#stopPartitions()`:
    a. Removes the scheduled tasks for the topic-partition from the thread pool responsible for archiving logs to remote storage.
    b. If the disablement policy is set to `delete`, the Log start offset (LSO) is updated to match the Local Log Start Offset and the remote log is deleted by calling the `RemoteStorageManager#deleteLogSegmentData()`.
    c. If the disablement policy is set to `retain`, there is no extra call required to delete remote data as the RemoteLogManager leader partition task will periodically expire segments that are earlier than the log start offset.
5. Controller interaction:
    a. The controller enqueues the disablement intent for tiered storage to a local queue.
    b. It sends an API call, called `DisableRemoteTopic`, to each broker with a callback to disable the remote topic.
    c. `DisableRemoteTopic` is a new controller-to-broker API that will be introduced in this KIP.
    d. Note that the current approach for communication between the Controller and brokers involves a familiar pattern. Tasks are enqueued in a local queue within the Controller Context, and a request is sent to the brokers along with a callback mechanism. This method is currently employed by the controller for operations, such as deleting topics. For instance, when the controller initiates the deletion of topics, it maintains a local queue called `topicsToBeDeleted`. This queue serves the purpose of monitoring the progress of topic deletions.
6. Completion of disablement:
    a. Once the controller receives responses from all brokers for the `DisableRemoteTopic` request, it updates the `tiered_state` in the TopicMetadata to "DISABLED".
    b. This update marks the completion of the disablement process, indicating that tiered storage has been successfully disabled for the Kraft-based clusters.

Failure Modes

- Controller Failover in DISABLING State: Following a new controller election, the controller context will be reconstructed for all topic-partitions that are in the DISABLING state. Additionally, the controller will initiate a new request for DisableRemoteTopic to all brokers.

- Broker Dies in DISABLING State or Fails to Complete the DisableRemoteTopic Request: The controller maintains an internal queue to track completed DisableRemoteTopic calls. In the event of a leader failover during this state, the controller will retry the operation until it receives a successful response. This behavior mirrors how we currently handle topic deletion.

Internal Interface Changes

1. `DisableRemoteTopic` is a new controller-to-broker API with the following schema. Note that this will bump the `inter.broker.protocol.version`.

DisableRemoteTopic

```
{
  "apiKey": TBD:(INT16),
  "type": "metadata",
  "name": "DisableRemoteTopic",
  "validVersions": "0",
  "flexibleVersions": "0+",
  "fields": [
    { "name": "PartitionId", "type": "int32", "versions": "0+", "default": "-1",
      "about": "The partition id." },
    { "name": "TopicId", "type": "uuid", "versions": "0+",
      "about": "The unique ID of this topic." }
  ]
}
```

2. The `RemoteLogSegmentMetadata` will have a new field, `tieredEpoch`.

```
public RemoteLogSegmentMetadata(RemoteLogSegmentId remoteLogSegmentId,
                                long startOffset,
                                long endOffset,
                                long maxTimestamp,
                                int leaderEpoch,
                                long eventTimestamp,
                                int segmentSizeInBytes,
                                RemoteLogSegmentState state,
                                Map<Int, Long> segmentLeaderEpochs,
                                int tieredEpoch) {
        this.remoteLogSegmentId = remoteLogSegmentId;
        this.startOffset = startOffset;
        this.endOffset = endOffset;
        this.leaderEpoch = leaderEpoch;
        this.maxTimestamp = maxTimestamp;
        this.eventTimestamp = eventTimestamp;
        this.segmentLeaderEpochs = segmentLeaderEpochs;
        this.state = state;
        this.segmentSizeInBytes = segmentSizeInBytes;
+       this.tieredEpoch = tieredEpoch;
    }
```

3. In ZooKeeper-based clusters, the topics' ZNode will have two new leaf ZNodes for storing the tiered epoch and the tiering state.

```
/brokers/topics/{topic-name}/partitions
                            /tieredstorage/
                                         /tiered_epoch
                                         /state
```

4. In Kraft-based clusters, the `TopicMetadata` contains a new field. The addition of this field will bump the metadata version.

```
MetadataResponse#TopicMetadata. Add new field tiered_epoch.

TopicMetadata => topic_error_code topic is_internal partition_metadata
  topic_error_code => int16
  topic => str
  is_internal => boolean
  partition_metadata => [PartitionMetadata]
+ tiered_epoch => int32
+ tiered_state => [TieredState]
```

Public Interfaces

Client API Changes

The `AlterConfigs` API will be updated to accommodate the following changes:

- Updating the topic-level configuration `remote.storage.enable` with a new configuration value of `false` will disable tiered storage.
- The `AlterConfigs` API will also be updated to support an optional tiered storage disablement policy, `remote.log.disable.policy=retain|delete`. If the disablement policy is not specified, it will default to `Retain`the remote logs.
- Re-enablement after disablement will not introduce any changes to the public interface.
- Re-enablement will not provide an option to specify a policy.

Exceptions

There are certain exceptional scenarios to consider:

- If a new disablement request is encountered while the topic is still in the `DISABLING` state, an exception named `TIERED_STORAGET_DISABLEMENT_IN_PROGRESS` will be thrown. This exception will have the following details:
  - Error: TIERED_STORAGET_DISABLEMENT_IN_PROGRESS
  - Code: > 71 (To be determined during implementation time)
- For other invalid formatted requests, such as an unsupported disablement policy option, the existing error `INVALID_REQUEST (42)` will be returned with an appropriate error message.

Configuration

A new configuration to support the disablement policy will be introduced.

```
Configuration: remote.log.disable.policy
Description: Determines whether tiered data for a topic should be retained or deleted after tiered storage
disablement on a topic.
Type: String
Default: retain
Valid values: retain, delete
Scope: topic wide
```

The configuration setting `remote.log.disable.policy` can also be introduced as a broker-level configuration, with the default value set to `retain`. For the initial implementation, we are suggesting this to be a topic level configuration, but we have the flexibility to make it to a broker-level configuration in future iterations.

```
# Defaults to remote.log.disable.policy=retain
bin/kafka-configs.sh --bootstrap-server {bootstrap-string} \
   --alter --entity-type topics --entity-name {topic-name} \
   --add-config 'remote.storage.enable=false'

#Disable with remote.log.disable.policy=retain
bin/kafka-configs.sh --bootstrap-server {bootstrap-string} \
   --alter --entity-type topics --entity-name {topic-name} \
   --add-config 'remote.log.disable.policy=retain, remote.storage.enable=false'

#Disable with remote.log.disable.policy=delete
bin/kafka-configs.sh --bootstrap-server {bootstrap-string} \
   --alter --entity-type topics --entity-name {topic-name} \
   --add-config 'remote.log.disable.policy=delete, remote.storage.enable=false'
```

Metrics

Further details to follow as the design progresses.

Compatibility, Deprecation, and Migration Plan

- This feature will be compatible with all Kafka versions that support Tiered Storage.
- This feature will be compatible with both Kraft-based and ZooKeeper-based clusters.

Test Plan

The test plan for this feature includes the following:

- Integration Tests:For integration tests, we will utilize a file-based tiering (LocalTieredStorage) to test the disablement code path.
- Unit Tests: Comprehensive unit test coverage will be provided to ensure the functionality of individual components involved in the disablement process.
- System Tests: System tests will be conducted for both ZooKeeper-based and Kraft-based clusters.

Rejected Alternatives

The following alternatives were considered but ultimately rejected:

1. Introducing a new API from the broker to the controller mode to notify the controller when the disablement process is completed. This alternative was not pursued as it represents a shift in the paradigm, where a broker initiates a call to the controller. It was deemed less suitable for the desired design approach.
2. Reuse the existing API `StopReplicaRequest` instead of creating a new API `DisableRemoteTopic` to notify brokers the disablement of tiered storage on a topic. This approach would function from a technical standpoint by simply adding an additional flag to indicate that the StopReplicaRequest is for the disablement of tiered storage. However, for the sake of clarity in intent, it is recommended to opt for the creation of the new API, DisableRemoteTopic. This choice ensures clear and unambiguous communication of the operation's purpose.
3. Introducing a temporary ZNode in `/{$AdminZNode.path}/remote_storage_topic_disablements/{$topic}` in ZooKeeper (ZK) mode to serve as a lock while the disablement process is in progress. This alternative was rejected because maintaining the state within the Topics ZNode is more intuitive and consolidates topic-related state in a single location.