KIP-986: Cross-Cluster Replication

Note that this proposal is incomplete, and tries to explore the UX of the feature before establishing the technical requirements and limitations. As we discover what the technical limitations are, some of the UX may need to change, and some semantics of the feature may need to be softened.

To Co-Authors: None of the below contents is final or necessarily correct. Please feel free to edit this document directly and summarize your changes on the mailing list afterwards.

Status

Current state: Under Discussion

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

Since before the implementation of MirrorMaker 1, there has been a desire to replicate data between Kafka clusters. This can be done for a many different reasons including but not limited to: disaster recovery, latency optimization, load shaping, security, and data protection. Currently the open source tooling for Kafka replication consists of MirrorMaker 1 and MirrorMaker 2, which both fall short in many modern use-cases.

1. They run in external processes which may experience outages when both Kafka clusters are otherwise healthy and accepting clients
2. They represent an additional operational burden beyond just running Kafka
3. Replication does not preserve offsets of individual records
4. Replication does not preserve exactly-once-semantics for records & consumer offsets

Goals

- Replicate topics and other associated resources between intentionally separate Kafka clusters
- Offsets for replicated records should be the same as in the origin Kafka
- Preserve exactly-once-semantics for replicated records

Non-Goals

- Implement multi-leader partitions that accept writes on multiple brokers.
- Perform distributed consensus to elect partition leaders across multiple clusters
- Provide transparent failover of single clients between Kafka clusters

Public Interfaces

User Interface

- New AdminClient methods for managing Replication Links on both the source and destination clusters
- A single cluster can participate in arbitrarily many links, and both a source and destination simultaneously.
- Links accept a configuration, including topics.regex and consumer.groups.regex, and a mode selector that accepts either Asynchronous or Synchronous
    - *Can a link change it's topics.regex or consumer.groups.regex? What happens if a topic or group is created or deleted while the link is in-sync? Does the link change state?*
    - *Can a link change from synchronous to asynchronous?*
- Existing Consumers & Producers can access replicated topics without an upgrade or reconfiguration.
- Metrics allow observation of the state of each partition included in the replication link and progress of the replication flow (lag, throughput, etc)
- During and shortly after network partitions, the link will be out-of-sync while it catches up with the source ISR
- Links can be manually disconnected, after which the destination topic becomes writable.
    - *Can we manually reverse a link while keeping consistency?*
    - *Can we reconnect a link, truncating the target if it has diverged?*

Data Semantics

Relation to Intra-Cluster replication

Cross-Cluster Replication is similar to Intra-Cluster replication, as both cross-cluster topics and intra-cluster replicas:

- Have the same configuration as their source
- Have the same offsets for records
- Have the same number of partitions

They are different in the following ways, as cross-cluster replicas:

- Are subject to the target cluster's ACL environment
- Are not eligible for fetches from source cluster consumers
- Have a separate topic-id

Replication Mode

- Asynchronous mode allows replication of a record to proceed after the source cluster has ack'd the record to the producer.
    - Maintains existing latency of source producers, but provides only single-topic consistency guarantees
- Synchronous mode requires the source cluster to delay acks (*or delay commits?*) for a producer until after the record has been replicated to all ISRs for all attached synchronous replication links.
    - Also applies to consumer group offsets submitted by the consumer or transactional producer
    - Increases latency of source producers, in exchange for multi-topic & consumer-offset consistency guarantees

Network Partition Behavior

We must support network partition tolerance, which requires choosing between consistency and availability. Availability in the table below means that the specified client can operate, at the expense of consistency with another client. Consistency means that the specified client will be unavailable, in order to be consistent with another client.

Mode	**Asynchronous Mode**			**Synchronous Mode**		
Link State	**Startup**	**Out-of-sync**	**Disconnected**	**Startup**	**Out-of-sync**	**Disconnected**
Source Consumers	Available[1]	Available[1]	Available[1,2]	Available[1]	Available[1]	Available[1,2]
Source Non-Transactional Producers	Available[1]	Available[1]	Available[1,2]	Available[1]	Available[1]	Available[1,2]
Source Transactional Producers	Available[3]	Available[4]	Available[2]	Available[3]	Consistent[5]	Available[2]
Target Consumers in Non-Replicated Group	Available[6]	Available[6]	Available[2]	Consistent[7]	Consistent[8]	Available[2]
Target Consumers in Replicated Group	Consistent[9]	Consistent[9]	Available[2]	Consistent[7,9]	Consistent[9]	Available[2]
Target Producers	Consistent[10]	Consistent[10]	Available[2]	Consistent[10]	Consistent[10]	Available[2]

1. Source clients not involved in transactions will always prioritize availability over consistency
2. When the link is permanently disconnected, clients on each cluster are not required to be consistent and can show whatever the state was when the link was disconnected.
3. Transactional clients are available during link startup, to allow creating links on transactional topics without downtime.
4. Transactional clients are available when asynchronous links are out-of-sync, because the source cluster is allowed to ack transactional produces while async replication is offline or behind.
5. Transactional clients are consistent when synchronous links are out-of-sync, because the destination topic is readable by target consumers. Transactional produces to a topic with an out-of-sync synchronous replication link should timeout or fail.
6. Consumers of an asynchronous replicated topic will see partial states while the link is catching up
7. While starting up a synchronous replicated topic, consumers will be unable to read the partial topic. This allows source transactional produces to proceed while the link is starting (3)
8. Consumers of a synchronous replicated topic should always see the same contents as the source topic

9. Consumers within a replicated consumer group will be unable to commit offsets, as the offsets are read-only while the link is active.
10. Producers targeting the replicated topic will fail because the topic is not writable until it is disconnected. Allowing a parallel-write to the replicated topic would be inconsistent.

Unclean Link Recovery

After a link is disconnected, the history of the two topics is allowed to diverge arbitrarily, as each leader will accept writes which are uncorrelated. There should be a way to reconnect a link, and reconcile the two histories by choosing one to overwrite the other.

This process can be manually-initiated, and involves choosing which direction the link should flow, and what the last shared offset was before the divergence occurred. Topics on the destination will be truncated to the last shared offset, and cross-cluster-replication will be restarted.

For a single link, we can remember what the last offset received by the target was prior to the disconnect, and truncate from there.

For example, a link AB is set up, A and B are partitioned, and the link AB is disconnected, and then the A/B network partition resolves. A and B diverge, and the operator chooses B as the new "source". A link BA is set up, and A is truncated to the last common offset before replication starts.

For multiple links, and links that didn't carry traffic, the last common offset depends on where the replicated data came from. Perhaps we can remember the provenance (cluster-id/topic-id) of ranges of offsets, and find the most recent offset which originated in the same topic amongst the clusters? Or force the operator/management layer to decide the truncation offset. Anything other than comparing hashes of messages or byte-for-byte equality checks.

For example two links AB and AC are set up, A is partitioned from B and C, B is chosen as the new leader, the links AB and AC should be disconnected, and a link BC connected. B and C will then need to perform unclean link recovery, determining the last offset that they both got from A prior to the disconnect. C then truncates to that common offset, and begins replicating from B.

Remote and Local Logs

- Remote log replication should be controlled by the second tier's provider.
- Remote logs can be referenced if not replicated by the second tier's provider so that replicated Kafka topics reference the source remote log storage.

Namespace Reservation

When a link is established, or reconfigured, it will reserve topic and consumer group namespaces in the destination cluster. These reservations will prevent coincidentally named topics and consumer groups from being created which would conflict with topics/consumer groups that could be created later in the source cluster.

- When a client tries to create a resource which has a namespace reservation, creating that resource should fail (*with a new error? with the most closely related existing error?*)
- When a link is established or reconfigured, and the necessary reservation conflicts with existing topics or existing reservations, those resources will become managed by the replication link.
- If a link is reconfigured such that a managed topic is no longer included, that topic will become disconnected, similar to if the link was disconnected for all topics.

Networking

- The network path between Kafka clusters is assumed to have less uptime, lower bandwidth, and higher latency than the intra-cluster network, and have more strict routing constraints.
- Network connections for replication links can proceed either sourcetarget or targetsource to allow one cluster to be behind a NAT *(can we support NAT punching as a first-class feature, to allow both clusters to be behind NAT?)*
- Allow for simple traffic separation of cross-cluster connections and client/intra-cluster connections *(do we need to connect to something other than the other cluster's advertised listeners?)*

Briefly list any new interfaces that will be introduced as part of this proposal or any existing interfaces that will be removed or changed. The purpose of this section is to concisely call out the public contract that will come along with this feature.

A public interface is any change to the following:

- *Binary log format*
- *The network protocol and api behavior*
- Any class in the public packages under clientsConfiguration, especially client configuration
    - org/apache/kafka/common/serialization
    - org/apache/kafka/common
    - org/apache/kafka/common/errors
    - org/apache/kafka/clients/producer
    - org/apache/kafka/clients/consumer (eventually, once stable)
- *Monitoring*
- *Command line tools and arguments*
- *Anything else that will likely break existing users in some way when they upgrade*

User Stories

Disaster Recovery (multi-zone Asynchronous)

1. I administrate multiple Kafka clusters in different availability zones
2. I have a performance-sensitive application that reads in all zones but writes to only one zone at a time. For example, an application that runs consumers in zones A and B to keep caches warm but disables producers in zone B while zone A is running.
3. I set up an asynchronous Cross-Cluster replication link for my topics and consumer groups from cluster A to cluster B. While the link is being created, applications in zone A are performant, and zone B can warm it's caches with historical data as it is replicated.
4. I do the same with cluster A and cluster C (and others that may exist)
5. When zone A goes offline, I want the application in zone B to start writing. I manually disconnect the AB cross-cluster link, and trigger the application in zone B to begin writing.
6. *What happens to cluster C? Can we connect BC quickly? What happens if C is ahead of B and truncating C breaks the cluster C consumers?*
7. When zone A recovers, I see that the history has diverged between zone A and B. I manually delete the topics in zone A and re-create the replication link in the opposite direction.

Proposed Changes

Describe the new thing you want to do in appropriate detail. This may be fairly extensive and have large subsections of its own. Or it may be a few sentences. Use judgement based on the scope of the change.

Compatibility, Deprecation, and Migration Plan

- Both the source and target clusters should have a version which includes the Cross-Cluster Replication feature
- Clusters which support the Cross-Cluster Replication feature should negotiate on the mutually-supported replication semantics
- If one of the clusters is downgraded to a version which does not support Cross-Cluster Replication, the partner cluster's link should fall out-of-sync.

Test Plan

Describe in few sentences how the KIP will be tested. We are mostly interested in system tests (since unit-tests are specific to implementation details). How will we know that the implementation works as expected? How will we know nothing broke?

Rejected Alternatives

Propose improvements to MirrorMaker 2 or a new MirrorMaker 3

Mirror Maker's approach to using public clients to perform replication limits the guarantees that replication provides. In order to strengthen these guarantees, we would need to add capabilities to the public clients, or rely on internal interfaces, neither of which is desirable.

Establish mechanisms for improving "stretched clusters" that have a heterogeneous network between nodes, aka "rack awareness"

The use-case for a stretched cluster is different than cross-cluster replication, in that a stretched cluster shares ACLs, topic-ids, principals, secrets, etc. Cross-Cluster Replication is intended to be used across data protection domains, which currently require the use of distinct clusters.

Propose a layer above Kafka to provide virtual/transparent replication

This is currently possible to implement with the Kafka public APIs, but doesn't actually replicate the data. This makes it unsuitable for disaster recovery, latency optimization, and load-shaping use-cases where connectivity to the source topic/replicas may be lost.