# Kafka Data Replication Project

## Overview

This project focuses on enhancing Apache Kafka's [MirrorMaker 2](https://cwiki.apache.org/confluence/display/KAFKA/KIP-382%3A+MirrorMaker+2.0) for mission-critical data replication between primary cluster (PR) and disaster recovery (DR) cluster scenarios. You are expected to (1) build a synthetic data replication pipeline and (2) make changes to the [source code](https://github.com/apache/kafka/tree/4.0.0/connect/mirror/src/main/java/org/apache/kafka/connect/mirror) of MirrorMaker 2 to enhance the fault tolerance aspect above what vanilla MirrorMaker 2 offers.

**Background**: This project simulates a real-world data replication pipeline where Kafka topics serve as Write-Ahead Logs (WAL) containing ordered events that represent state changes in a distributed system.

In the primary cluster, applications produce events to the commit-log topic, which acts as the authoritative record of all system operations. These events are continuously consumed and replayed by services to maintain their local state. Simultaneously, we must replicate this event stream to a disaster recovery (DR) cluster to ensure business continuity.

The replication pipeline works as follows:

1. **Primary Cluster**: Events are written to commit-log and replayed by local services
2. **Cross-Cluster Replication**: MirrorMaker 2 replicates events from commit-log to primary.commit-log in the DR cluster
3. **DR Cluster**: Standby services consume from primary.commit-log to maintain a synchronized state

However, MirrorMaker 2's default behavior doesn't account for critical failure scenarios that can compromise this pipeline's reliability, particularly around data loss detection and recovery from infrastructure changes.

**Technical Challenge**: Real-world Kafka deployments face two critical scenarios that can compromise data integrity:

1. **Silent Data Loss**: Kafka retention policies may purge data from source topics before replication completes, creating undetectable gaps in the replicated data stream.
2. **Service Disruption**: Planned maintenance operations involving topic reset(topic deletion and recreation) can cause the replication service to be unable to find the expected offset and stop replication.

**Project Objective**: Enhance MirrorMaker 2 with intelligent fault detection and automatic recovery capabilities to handle these scenarios gracefully. **You are required to fork the** [**Kafka repository**](https://github.com/apache/kafka/tree/4.0.0) **and modify the source code of MirrorMaker 2.**

**Complexity**: The estimated modifications to MirrorMaker 2's source code are under 500 LOC. The solution should integrate seamlessly with the existing codebase while providing comprehensive data loss detection and automatic recovery capabilities.

**Source Code**: Use Apache Kafka from <https://github.com/apache/kafka> (v4.0.0).

**AI Usage Policy**: If AI tools (Cursor, Copilot, Claude Code etc.) are used to accelerate development. We expect you to:

* Fully understand every line of code in your final deliverable
* Be able to articulate the design decisions and implementation details
* Document your AI usage methodology in the README.md
* Explain how AI helped you solve specific challenges

## Architecture

You are required to configure both the primary and standby clusters and to develop the two components outlined below.

### Cluster Setup

* **Primary Cluster**: Single-node Kafka cluster hosting commit-log topic
* **Standby Cluster**: Single-node Kafka cluster hosting primary.commit-log topic
* **Tip**: You can use the [Apache Kafka Docker image](https://hub.docker.com/r/apache/kafka) to set up the two clusters

### Topic Setup

* **Topic Configuration**: Both topics with exactly 1 partition, 1 replica
* **Test Configuration**: Set log.retention.ms=60000 (60 seconds) on commit-log for truncation testing

### Components

1. **Commit Log Producer**: CLI application generating JSON events to commit-log
2. **Enhanced MirrorMaker 2**: Modified MirrorMaker 2 service with error-handling features

### Event Schema

{

"event\_id": "a8a1c867-05c3-4d43-9884-f7b55f1f0a7c",

"timestamp": 1724684407,

"op\_type": "UPDATE",

"key": "doc:8f7b",

"value": {

"status": "archived"

}

}

## Core Tasks

### Task 1: Commit Log Producer

**Objective**: CLI application generating JSON events to the primary cluster's commit-log topic.

**Requirements**:

* Accept --count N parameter to produce exactly N messages then exit
* Generate valid JSON with unique UUIDs and current timestamps

### Task 2: Log Truncation Detection (Fail-Fast)

**Problem**: Aggressive retention policies may purge messages before replication, causing silent data loss.

**Solution**: Enhance MirrorMaker 2 to detect and respond to log truncation.

**Technical Requirements**:

* Detect offset gaps indicating truncated data
* Upon detection: log detailed error, and throw an exception to fail-fast
* Integration: Minimal disruption to existing MirrorMaker 2 logic

### Task 3: Graceful Topic Reset Handling

**Problem**: Topic deletion/recreation can cause MirrorMaker 2 failures or stalls.

**Solution**: Add automatic recovery capabilities to MirrorMaker 2.

**Technical Requirements**:

* Detect source topic reset(topic deletion and recreation)
* Log reset events with timestamp and topic details
* Automatically resubscribe from the beginning offset

## Deliverables

### 1. Source Code

**Required Repository**:

* **Kafka Fork**: Fork <https://github.com/apache/kafka> and create a pull request with MirrorMaker 2 modifications

**Docker Hub Images**: Push the following custom Docker images to your Docker Hub account:

* **Enhanced MirrorMaker 2**: Pre-built Kafka with your fault-tolerance modifications
* **Commit Log Producer**: CLI application for generating test events

### 2. Docker Compose Setup

**docker-compose.yml**: Complete environment setup including:

* Primary Kafka cluster
* Standby Kafka cluster
* Enhanced MirrorMaker 2 (using your custom image)
* Commit Log Producer

### 3. Automation Scripts

**run\_challenge.sh**: Orchestrate test scenarios using Docker Compose:

* **Normal replication flow**: Producer generates 1000 messages, verifies replication
* **Log truncation simulation**: Trigger truncation, verify MirrorMaker 2 detects and fails
  + **Tip**: Consider pausing the MirrorMaker 2 service until the Primary cluster reaches its topic retention time.
* **Topic reset simulation**: Delete/recreate topic, verify MirrorMaker 2 recovers automatically
  + **Tip**: Consider pausing the MirrorMaker 2 service until the Primary cluster recreates the commit-log topic.

### 4. Documentation

**README.md** must include:

* **Repository Links**: Links to Kafka fork and pull request
* **Docker Hub Images**: List of published images with tags
* **Setup Instructions**: docker-compose up and initial configuration
* **Test Execution**: How to run run\_challenge.sh and interpret results
* **Log Analysis**: Key log messages and container logs to monitor
* **Design Rationale**: Explanation of MirrorMaker 2 modifications and integration approach
* **AI Usage Documentation**: If AI tools were used, detail the methodology, specific prompts/queries, and how AI contributed to the solution

### 5. Code Quality Standards

* **Production-Ready**: Clean, efficient, well-structured code
* **Modular Design**: Minimal disruption to the existing Kafka codebase
* **Comprehensive Logging**: Clear error messages using SLF4J
* **Pull Request Quality**: Clear commit messages, proper PR description explaining changes

## Grading Criteria

* **Functional Correctness**
  + All three tasks are implemented and working correctly
  + Proper handling of normal replication, truncation, and reset scenarios
  + Data integrity is maintained across all test cases
* **Technical Excellence**
  + Deep understanding of Kafka internals and distributed systems principles
  + Efficient, maintainable code with minimal impact on the existing codebase
  + Proper error handling and logging mechanisms
* **Documentation & Automation**
  + Complete, clear documentation with setup and execution instructions
  + Robust automation scripts that reliably demonstrate all scenarios
  + Comprehensive log analysis guidance