Confluent Replicator: Introduction & Key Capabilities

Reliable Cross-Cluster Kafka Replication for Enterprise Environments

What is Confluent Replicator
An enterprise-grade Kafka Connect connector that replicates topics, configurations, and schemas between Kafka clusters across data centers or clouds.

Core Capabilities
Provides topic replication, configuration preservation, offset translation, schema synchronization, circular replication prevention, and multi-datacenter topology support.

When to Use
Ideal for disaster recovery, geo-localized deployments, cloud migration, centralized analytics, compliance archiving, and workload distribution.

Enterprise Reliability
Backed by Confluent’s managed support, integrates seamlessly with Confluent Platform and Control Center for monitoring and automation.

Architecture & Core Concepts

How Confluent Replicator Enables Reliable Kafka Data Movement

- **Core Components:** Includes source and destination clusters, Replicator connector, and Kafka Connect workers that manage task execution, configuration storage, and offset management.
- **Replication Workflow:** Consumes messages from source partitions, maps offsets by timestamp, and produces to destination topics with provenance headers to prevent circular replication.
- **Design Principles:** One-directional by default, bidirectional via multiple instances, maintains partition fidelity, ensures offset tracking and recovery after interruptions.
- **Operational Integration:** Deployed within Kafka Connect clusters for scalability and monitoring via Control Center or JMX; supports both standalone and distributed modes.

Deployment Models

Choosing the Right Architecture for Confluent Replicator

- **Deployment Near Destination Cluster:** Recommended approach for production; minimizes latency and failure impact by placing Replicator close to the target Kafka cluster where writes occur.
- **Standalone Replicator:** Single instance suitable for small-scale or test environments; simple to configure but limited scalability and fault tolerance.
- **Dedicated Connect Cluster:** Runs Replicator on its own Connect cluster, isolating resources and optimizing for replication workloads; ideal for production-critical replication.
- **Shared Connect Cluster:** Replicator runs alongside other connectors in the same Connect cluster; simpler infrastructure but risks resource contention and reduced performance.

Configuration & Setup

Essential Properties and Execution Modes for Replicator

Core Configuration Files

Replicator executable requires three property files: consumer.properties for source, producer.properties for destination, and replication.properties for Connect environment.

Connector JSON Configuration

When running within a Connect cluster, configurations define connector class, source/destination brokers, consumer group, topic selection, and task parallelism.

Essential Parameters

Key settings include bootstrap servers, group.id, tasks.max, and topic selection. Advanced options cover renaming, compression, and converter formats.

Execution & Verification

Replicator can be launched as standalone executable or via REST API; use Connect status endpoints to validate running state and monitor tasks.

Topic Selection & Replication Strategies

Controlling Which Kafka Topics Are Replicated

- **Selection Mechanisms:** Replicator supports whitelist, regex, and blacklist methods to define topics to include or exclude, enabling precise control of replication scope.
- **Selection Logic Order:** Replication order: start with all topics → apply whitelist → add regex matches → remove blacklist topics; result defines final replication set.
- **Dynamic Topic Discovery:** Automatically detects and replicates new topics matching defined patterns without manual reconfiguration, using configurable polling intervals.
- **Topic Renaming:** Supports renaming during replication for namespacing, environment differentiation, and tenant isolation using rename.topics property.

Security Configuration

Securing Cross-Cluster Replication with Authentication and ACLs

- **Authentication Mechanisms:** Supports SASL/SCRAM for secure password-based authentication, SASL/PLAIN for simpler setups, and SSL/TLS for certificate-based trust establishment.
- **Authorization via ACLs:** Grants read access to source topics and write/create permissions on destination topics; consumer group ACLs manage offset synchronization.
- **Network Security:** Uses VPC peering and firewall rules to restrict connectivity; only Replicator hosts can reach Kafka brokers on secured ports (9092/9093).
- **Best Practices:** Always use SASL_SSL with SCRAM-SHA-256, rotate credentials periodically, and exclude internal topics to prevent unauthorized replication.

Schema Registry Integration

Ensuring Schema Consistency Across Replicated Topics

Automatic Schema Synchronization
Replicator can replicate Avro, Protobuf, and JSON schemas between source and destination Schema Registries, ensuring schema evolution consistency.

Configuration Requirements
Requires specifying both src and dest Schema Registry URLs, credentials, and converters (KafkaAvroSerializer, KafkaProtobufSerializer, or KafkaJsonSchemaSerializer).

Schema Subject Translation
Renames schema subjects when topics are renamed during replication using schema.subject.translator.class and rename.topics parameters.

Compatibility Management
Preserves backward, forward, and full schema compatibility, enabling seamless consumer interoperability after replication.

Offset Translation & Consumer Group Migration

Seamless Consumer Failover Between Clusters

- **Purpose of Offset Translation:** Maps consumer offsets from source to destination cluster using timestamp-based mapping, allowing consumers to resume processing without re-reading data.
- **Mechanism:** Replicator records message timestamps, replicates them to destination, and translates offsets by locating matching timestamps, ensuring consistent read positions.
- **Configuration:** Enable via offset.translator.enabled=true; requires monitoring consumer interceptor on source cluster to capture commit timestamps.
- **Failover Workflow:** During DR or migration, stop consumers on source, let Replicator catch up, and restart consumers on destination using translated offsets for continuity.

Monitoring & Metrics

Observability and Health Tracking for Confluent Replicator

Core Metrics
Key indicators include throughput (records-per-sec, bytes-per-sec), message lag, latency, retry rate, and error rate for replication health.

Lag & Latency Analysis
Lag measures pending messages; latency tracks replication delay. Typical values: <1s intra-DC, <2s cross-DC; increasing lag signals performance issues.

Monitoring Tools
Confluent Control Center provides task-level insights, throughput graphs, and alerting; REST API and JMX support external integrations.

Alerting & Automation
Set thresholds for lag, latency, and error rates; automate alerts when throughput drops or replication stalls for proactive recovery.

Performance Tuning

Optimizing Throughput, Latency, and Resource Utilization

- **Baseline Measurement:** Establish baseline throughput (MBps, records/sec) per task before scaling. Example: 30 MBps/task to plan capacity accurately.
- **Scaling Tasks & Workers:** Increase tasks.max and distribute across Connect workers for horizontal scalability and fault tolerance; scale linearly with resources.
- **Consumer & Producer Tuning:** Adjust fetch.min.bytes, max.poll.records, batch.size, and linger.ms for optimal batching and network efficiency under high-latency conditions.
- **Compression & GC Optimization:** Use Snappy or Zstd compression to reduce bandwidth; apply G1GC with 4–8 GB heap for low-latency performance and predictable GC pauses.

Troubleshooting & Best Practices

Ensuring Reliable Operation and Quick Recovery

- **Common Issues:** Frequent problems include task failures, increasing lag, authentication errors, and offset translation issues due to misconfiguration or connectivity loss.
- **Diagnostic Workflow:** Use Connect REST API and worker logs to identify failing tasks; examine throughput, lag, and error metrics to locate bottlenecks.
- **Preventing Circular Replication:** Avoid infinite loops by enabling provenance headers and excluding internal topics (topic.blacklist=__.*).
- **Operational Best Practices:** Deploy dedicated clusters for replication, monitor continuously, test failovers, rotate credentials, and document topology and security configurations.

Real-World Use Cases

How Enterprises Apply Confluent Replicator in Production

Disaster Recovery (DR)
Maintains a standby Kafka cluster synchronized with the primary for rapid failover; ensures business continuity with minimal data loss.

Geo-Distributed Deployments
Replicates data across multiple active regions for locality-based consumption, enabling low-latency access and regional independence.

Hub-and-Spoke Architecture
Centralized data ingestion replicated to multiple edge clusters for regional processing or analytics, simplifying data distribution.

Cloud Migration
Facilitates seamless transition from on-premises Kafka to Confluent Cloud, maintaining real-time synchronization during migration phases.

Replicator vs. MirrorMaker 2 Comparison

Feature and Deployment Differences

Licensing & Support
Replicator is a commercial, enterprise-supported tool within Confluent Platform, while MirrorMaker 2 is open-source and community-maintained.

Feature Parity
Replicator offers built-in offset translation, schema replication, Control Center integration, and enterprise-grade monitoring; MM2 covers basic replication.

Operational Complexity
Replicator simplifies configuration with integrated Confluent ecosystem support; MM2 requires manual setup and lacks schema synchronization.

Use Case Alignment
Choose Replicator for Confluent-managed deployments or schema-heavy use cases; choose MM2 for cost-sensitive, open-source environments.

Key Takeaways & Resources

Summary of Concepts and Further Learning

Core Concepts

Confluent Replicator ensures reliable, schema-aware cross-cluster Kafka replication with offset translation and circular replication protection.

Deployment & Operations

Best practice: deploy near destination clusters, use dedicated Connect clusters, monitor continuously, and tune for throughput and resilience.

Real-World Impact

Enables disaster recovery, multi-region replication, and hybrid cloud migration with enterprise-grade monitoring and governance.

Further Resources

Consult Confluent documentation, training (CCDAK, CCO), and community channels for continued learning and certification.