Kafka Connect: Complete Developer Guide

A comprehensive refresher on Apache Kafka Connect, designed for both beginners and experienced developers. This README covers connectors, deployment modes, configuration, customization, and real-world applications with detailed Java examples.

Table of Contents

- Basics
  - Source vs Sink Connectors
  - Standalone vs Distributed Mode
  - Config Management via REST API
- Common Connectors
  - JDBC Source/Sink
  - Debezium CDC
  - Elasticsearch/S3 Connectors
- Customization
  - Writing Custom Connectors
  - Single Message Transforms (SMTs)
- Comprehensive Java Examples
- Comparisons & Trade-offs
- Common Pitfalls & Best Practices
- Real-World Use Cases
- Version Highlights
- Additional Resources

Basics

Simple Explanation

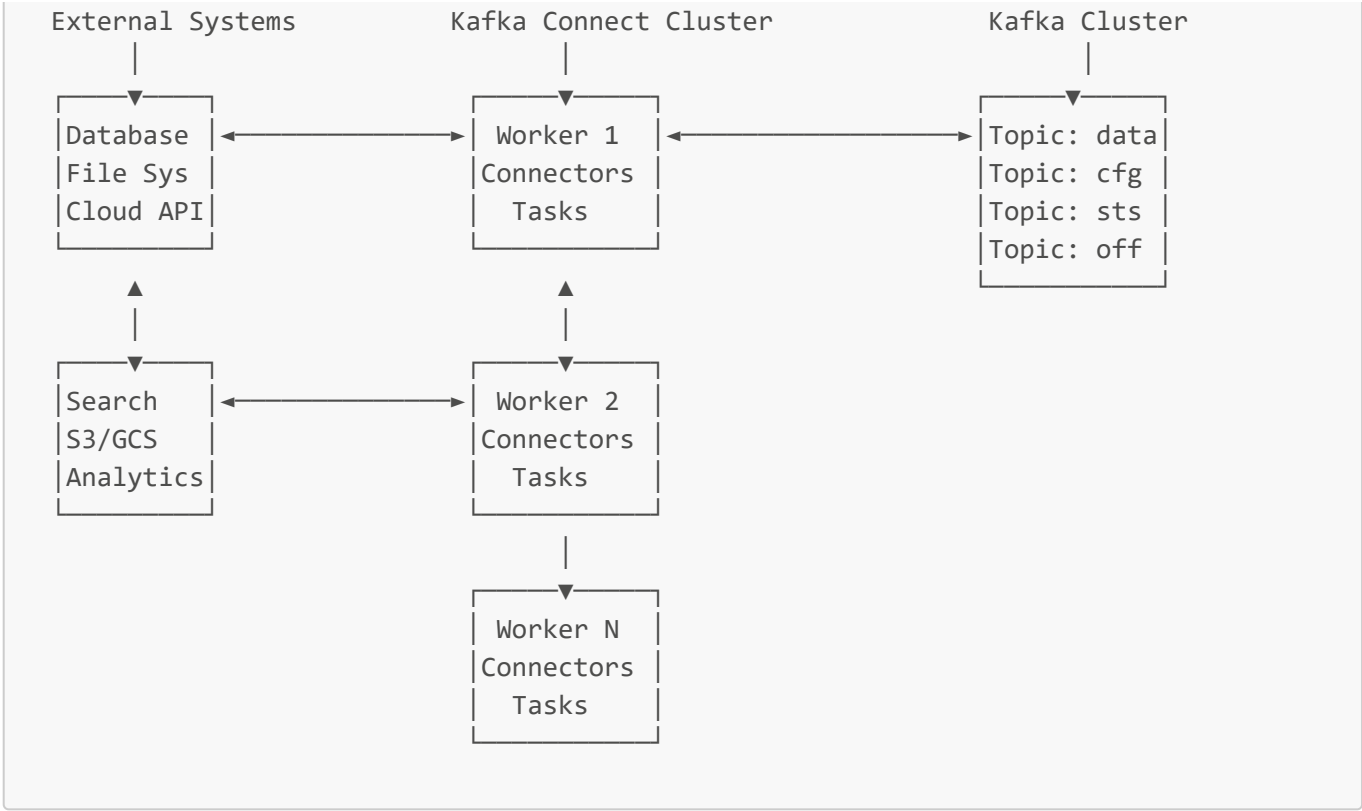
Kafka Connect is a framework for building and running reusable connectors that move data between Apache Kafka and other systems. It provides a REST API for managing connectors, automatic offset management, distributed operation, and fault tolerance.

Problem It Solves

- **Data Integration**: Eliminates custom code for moving data in/out of Kafka
- **Scalability**: Distributes connector tasks across multiple workers
- **Fault Tolerance**: Automatic recovery and exactly-once delivery guarantees
- **Operational Simplicity**: Centralized management via REST API

Internal Architecture

```
Kafka Connect Architecture:
```



How It Works Under the Hood

1. **Workers**: JVM processes that execute connector code and manage tasks
2. **Connectors**: Define how to copy data (splitting into tasks)
3. **Tasks**: Actual units of work that copy data
4. **Offset Storage**: Tracks progress in external systems and Kafka
5. **Configuration Storage**: Stores connector configurations in Kafka topics

Relevant Data Structures

- **connect-configs**: Compacted topic storing connector configurations
- **connect-offsets**: Tracks source connector progress (external system offsets)
- **connect-status**: Monitors connector and task status

Source vs Sink Connectors

Source Connectors - Import Data INTO Kafka

```
// Example: Database Source Connector Flow
External Database → Source Connector → Kafka Topic

// Flow:
1. Poll external system for new data
2. Convert to Kafka records
3. Send to Kafka topics
4. Track offsets in external system
```

Sink Connectors - Export Data FROM Kafka

```
// Example: Elasticsearch Sink Connector Flow
Kafka Topic → Sink Connector → Elasticsearch Index

// Flow:
1. Consume records from Kafka
2. Transform to target format
3. Write to external system
4. Commit Kafka offsets
```

Source Connector Example Configuration

```
import java.util.Properties;
import java.util.Map;
import java.util.HashMap;

public class SourceConnectorConfig {

    public static Map<String, String> createJDBCSourceConfig() {
        Map<String, String> config = new HashMap<>();

        // Connector class
        config.put("connector.class",
"io.confluent.connect.jdbc.JdbcSourceConnector");

        // Connection properties
        config.put("connection.url", "jdbc:postgresql://localhost:5432/mydb");
        config.put("connection.user", "user");
        config.put("connection.password", "password");

        // Source configuration
        config.put("table.whitelist", "users,orders,products");
        config.put("mode", "incrementing");
        config.put("incrementing.column.name", "id");
        config.put("timestamp.column.name", "updated_at");

        // Topic naming
        config.put("topic.prefix", "postgres-");

        // Polling configuration
        config.put("poll.interval.ms", "5000");
        config.put("batch.max.rows", "100");

        return config;
    }
}
```

Sink Connector Example Configuration

```
public class SinkConnectorConfig {

    public static Map<String, String> createElasticsearchSinkConfig() {
        Map<String, String> config = new HashMap<>();

        // Connector class
        config.put("connector.class",
"io.confluent.connect.elasticsearch.ElasticsearchSinkConnector");

        // Connection properties
        config.put("connection.url", "http://localhost:9200");
        config.put("connection.username", "elastic");
        config.put("connection.password", "password");

        // Topics to consume
        config.put("topics", "postgres-users,postgres-orders");

        // Index configuration
        config.put("type.name", "_doc");
        config.put("key.ignore", "false");
        config.put("schema.ignore", "true");

        // Batching
        config.put("batch.size", "100");
        config.put("max.in.flight.requests", "5");

        // Error handling
        config.put("behavior.on.null.values", "ignore");
        config.put("behavior.on.malformed.documents", "warn");

        return config;
    }
}
```

Standalone vs Distributed Mode

Standalone Mode

```
public class StandaloneConnectExample {

    public static void main(String[] args) {
        // Standalone mode configuration
        Properties workerProps = new Properties();
        workerProps.put("bootstrap.servers", "localhost:9092");
        workerProps.put("key.converter",
"org.apache.kafka.connect.json.JsonConverter");
        workerProps.put("value.converter",
"org.apache.kafka.connect.json.JsonConverter");
```

```
        workerProps.put("key.converter.schemas.enable", false);
        workerProps.put("value.converter.schemas.enable", false);

        // Offset storage (file-based for standalone)
        workerProps.put("offset.storage.file.filename", "/tmp/connect.offsets");
        workerProps.put("offset.flush.interval.ms", 10000);

        // Plugin path
        workerProps.put("plugin.path", "/usr/share/java,/usr/share/confluent-hub-
components");

        System.out.println("Standalone Connect Configuration:");
        workerProps.forEach((key, value) ->
            System.out.println(key + "=" + value));
    }

    /*
     * Standalone Mode Characteristics:
     *
     * Pros:
     * - Simple setup and configuration
     * - Good for development and testing
     * - Single point of failure acceptable
     *
     * Cons:
     * - No fault tolerance
     * - Limited scalability
     * - Manual configuration management
     *
     * Use Cases:
     * - Development environments
     * - Simple integrations
     * - Proof of concepts
     */
}
```

Distributed Mode

```
public class DistributedConnectExample {

    public static void main(String[] args) {
        // Distributed mode configuration
        Properties workerProps = new Properties();
        workerProps.put("bootstrap.servers", "localhost:9092");

        // Group and client identification
        workerProps.put("group.id", "connect-cluster");
        workerProps.put("client.id", "connect-worker-1");

        // Converters
        workerProps.put("key.converter",
```

```
"org.apache.kafka.connect.json.JsonConverter");
        workerProps.put("value.converter",
"org.apache.kafka.connect.json.JsonConverter");
        workerProps.put("key.converter.schemas.enable", true);
        workerProps.put("value.converter.schemas.enable", true);

        // Internal topic configuration
        workerProps.put("config.storage.topic", "connect-configs");
        workerProps.put("config.storage.replication.factor", 3);

        workerProps.put("offset.storage.topic", "connect-offsets");
        workerProps.put("offset.storage.replication.factor", 3);
        workerProps.put("offset.storage.partitions", 25);

        workerProps.put("status.storage.topic", "connect-status");
        workerProps.put("status.storage.replication.factor", 3);
        workerProps.put("status.storage.partitions", 5);

        // Worker configuration
        workerProps.put("offset.flush.interval.ms", 10000);
        workerProps.put("task.shutdown.graceful.timeout.ms", 30000);

        // REST API
        workerProps.put("rest.port", 8083);
        workerProps.put("rest.host.name", "localhost");

        // Plugin path
        workerProps.put("plugin.path", "/usr/share/java,/usr/share/confluent-hub-
components");

        System.out.println("Distributed Connect Configuration:");
        workerProps.forEach((key, value) ->
            System.out.println(key + "=" + value));
    }

    /*
     * Distributed Mode Characteristics:
     *
     * Pros:
     * - Fault tolerant (automatic failover)
     * - Horizontally scalable
     * - Load balancing across workers
     * - REST API for management
     *
     * Cons:
     * - More complex setup
     * - Requires Kafka for coordination
     * - Network overhead
     *
     * Use Cases:
     * - Production environments
     * - High-availability requirements
     * - Large-scale data integration
```

```
     */
}
```

Config Management via REST API

REST API Client Example

```
import java.io.IOException;
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.time.Duration;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.JsonNode;

public class KafkaConnectRestClient {

    private final HttpClient httpClient;
    private final String connectUrl;
    private final ObjectMapper objectMapper;

    public KafkaConnectRestClient(String connectUrl) {
        this.connectUrl = connectUrl;
        this.httpClient = HttpClient.newBuilder()
            .connectTimeout(Duration.ofSeconds(10))
            .build();
        this.objectMapper = new ObjectMapper();
    }

    // List all connectors
    public String[] listConnectors() throws IOException, InterruptedException {
        HttpRequest request = HttpRequest.newBuilder()
            .uri(URI.create(connectUrl + "/connectors"))
            .GET()
            .build();

        HttpResponse<String> response = httpClient.send(request,
            HttpResponse.BodyHandlers.ofString());

        if (response.statusCode() == 200) {
            return objectMapper.readValue(response.body(), String[].class);
        } else {
            throw new RuntimeException("Failed to list connectors: " +
response.body());
        }
    }

    // Create or update connector
    public void createOrUpdateConnector(String name, Map<String, String> config)
            throws IOException, InterruptedException {
```

```
        Map<String, Object> connectorConfig = new HashMap<>();
        connectorConfig.put("name", name);
        connectorConfig.put("config", config);

        String requestBody = objectMapper.writeValueAsString(connectorConfig);

        HttpRequest request = HttpRequest.newBuilder()
            .uri(URI.create(connectUrl + "/connectors"))
            .header("Content-Type", "application/json")
            .POST(HttpRequest.BodyPublishers.ofString(requestBody))
            .build();

        HttpResponse<String> response = httpClient.send(request,
            HttpResponse.BodyHandlers.ofString());

        if (response.statusCode() != 201 && response.statusCode() != 200) {
            throw new RuntimeException("Failed to create connector: " +
response.body());
        }

        System.out.println("Connector created/updated successfully: " + name);
    }

    // Get connector status
    public ConnectorStatus getConnectorStatus(String name)
            throws IOException, InterruptedException {

        HttpRequest request = HttpRequest.newBuilder()
            .uri(URI.create(connectUrl + "/connectors/" + name + "/status"))
            .GET()
            .build();

        HttpResponse<String> response = httpClient.send(request,
            HttpResponse.BodyHandlers.ofString());

        if (response.statusCode() == 200) {
            JsonNode statusNode = objectMapper.readTree(response.body());
            return new ConnectorStatus(
                statusNode.get("name").asText(),
                statusNode.get("connector").get("state").asText(),
                statusNode.get("tasks").size()
            );
        } else {
            throw new RuntimeException("Failed to get connector status: " +
response.body());
        }
    }

    // Restart connector
    public void restartConnector(String name) throws IOException,
InterruptedException {
        HttpRequest request = HttpRequest.newBuilder()
            .uri(URI.create(connectUrl + "/connectors/" + name + "/restart"))
```

```
            .POST(HttpRequest.BodyPublishers.noBody())
            .build();

        HttpResponse<String> response = httpClient.send(request,
            HttpResponse.BodyHandlers.ofString());

        if (response.statusCode() != 204) {
            throw new RuntimeException("Failed to restart connector: " +
response.body());
        }

        System.out.println("Connector restarted successfully: " + name);
    }

    // Pause connector
    public void pauseConnector(String name) throws IOException,
InterruptedException {
        HttpRequest request = HttpRequest.newBuilder()
            .uri(URI.create(connectUrl + "/connectors/" + name + "/pause"))
            .PUT(HttpRequest.BodyPublishers.noBody())
            .build();

        HttpResponse<String> response = httpClient.send(request,
            HttpResponse.BodyHandlers.ofString());

        if (response.statusCode() != 202) {
            throw new RuntimeException("Failed to pause connector: " +
response.body());
        }

        System.out.println("Connector paused successfully: " + name);
    }

    // Resume connector
    public void resumeConnector(String name) throws IOException,
InterruptedException {
        HttpRequest request = HttpRequest.newBuilder()
            .uri(URI.create(connectUrl + "/connectors/" + name + "/resume"))
            .PUT(HttpRequest.BodyPublishers.noBody())
            .build();

        HttpResponse<String> response = httpClient.send(request,
            HttpResponse.BodyHandlers.ofString());

        if (response.statusCode() != 202) {
            throw new RuntimeException("Failed to resume connector: " +
response.body());
        }

        System.out.println("Connector resumed successfully: " + name);
    }

    // Delete connector
    public void deleteConnector(String name) throws IOException,
```

```
InterruptedException {
        HttpRequest request = HttpRequest.newBuilder()
            .uri(URI.create(connectUrl + "/connectors/" + name))
            .DELETE()
            .build();

        HttpResponse<String> response = httpClient.send(request,
            HttpResponse.BodyHandlers.ofString());

        if (response.statusCode() != 204) {
            throw new RuntimeException("Failed to delete connector: " +
response.body());
        }

        System.out.println("Connector deleted successfully: " + name);
    }

    // Get connector configuration
    public Map<String, String> getConnectorConfig(String name)
            throws IOException, InterruptedException {

        HttpRequest request = HttpRequest.newBuilder()
            .uri(URI.create(connectUrl + "/connectors/" + name + "/config"))
            .GET()
            .build();

        HttpResponse<String> response = httpClient.send(request,
            HttpResponse.BodyHandlers.ofString());

        if (response.statusCode() == 200) {
            return objectMapper.readValue(response.body(),
                objectMapper.getTypeFactory().constructMapType(Map.class,
String.class, String.class));
        } else {
            throw new RuntimeException("Failed to get connector config: " +
response.body());
        }
    }

    // Usage example
    public static void main(String[] args) {
        KafkaConnectRestClient client = new
KafkaConnectRestClient("http://localhost:8083");

        try {
            // List existing connectors
            String[] connectors = client.listConnectors();
            System.out.println("Existing connectors: " + String.join(", ",
connectors));

            // Create JDBC source connector
            Map<String, String> jdbcConfig =
SourceConnectorConfig.createJDBCSourceConfig();
            client.createOrUpdateConnector("postgres-source", jdbcConfig);
```

```
            // Check status
            ConnectorStatus status = client.getConnectorStatus("postgres-source");
            System.out.println("Connector status: " + status);

            // Create Elasticsearch sink connector
            Map<String, String> esConfig =
SinkConnectorConfig.createElasticsearchSinkConfig();
            client.createOrUpdateConnector("elasticsearch-sink", esConfig);

        } catch (Exception e) {
            System.err.println("Error managing connectors: " + e.getMessage());
            e.printStackTrace();
        }
    }

    // Helper class for connector status
    public static class ConnectorStatus {
        private final String name;
        private final String state;
        private final int taskCount;

        public ConnectorStatus(String name, String state, int taskCount) {
            this.name = name;
            this.state = state;
            this.taskCount = taskCount;
        }

        @Override
        public String toString() {
            return String.format("ConnectorStatus{name='%s', state='%s',
tasks=%d}",
                    name, state, taskCount);
        }

        // Getters
        public String getName() { return name; }
        public String getState() { return state; }
        public int getTaskCount() { return taskCount; }
    }
}
```

Common Connectors

JDBC Source/Sink

JDBC Source Connector - Database to Kafka

```
import java.util.Map;
import java.util.HashMap;
```

```
public class JDBCSourceConnectorExample {

    public static Map<String, String> createIncrementalSourceConfig() {
        Map<String, String> config = new HashMap<>();

        // Connector identification
        config.put("name", "postgres-users-source");
        config.put("connector.class",
"io.confluent.connect.jdbc.JdbcSourceConnector");

        // Database connection
        config.put("connection.url",
"jdbc:postgresql://localhost:5432/ecommerce");
        config.put("connection.user", "postgres");
        config.put("connection.password", "password");
        config.put("connection.attempts", "3");
        config.put("connection.backoff.ms", "10000");

        // Mode configuration
        config.put("mode", "incrementing");
        config.put("incrementing.column.name", "id");

        // Table configuration
        config.put("table.whitelist", "users");
        config.put("catalog.pattern", null);
        config.put("schema.pattern", "public");

        // Topic configuration
        config.put("topic.prefix", "postgres-");
        config.put("topic.creation.enable", "true");

        // Polling configuration
        config.put("poll.interval.ms", "5000");
        config.put("batch.max.rows", "100");
        config.put("table.poll.interval.ms", "60000");

        // Query customization
        config.put("query", "SELECT * FROM users WHERE id > ? ORDER BY id");
        config.put("validate.non.null", "false");

        return config;
    }

    public static Map<String, String> createTimestampSourceConfig() {
        Map<String, String> config = new HashMap<>();

        // Basic configuration
        config.put("name", "postgres-orders-source");
        config.put("connector.class",
"io.confluent.connect.jdbc.JdbcSourceConnector");

        // Connection
        config.put("connection.url",
```

```
"jdbc:postgresql://localhost:5432/ecommerce");
        config.put("connection.user", "postgres");
        config.put("connection.password", "password");

        // Timestamp mode for CDC
        config.put("mode", "timestamp+incrementing");
        config.put("timestamp.column.name", "updated_at");
        config.put("incrementing.column.name", "id");

        // Table configuration
        config.put("table.whitelist", "orders");
        config.put("topic.prefix", "postgres-");

        // Polling configuration
        config.put("poll.interval.ms", "1000");
        config.put("batch.max.rows", "500");

        // Timestamp configuration
        config.put("timestamp.delay.interval.ms", "1000");
        config.put("db.timezone", "UTC");

        return config;
    }

    // Custom query example
    public static Map<String, String> createCustomQuerySourceConfig() {
        Map<String, String> config = new HashMap<>();

        config.put("name", "postgres-analytics-source");
        config.put("connector.class",
"io.confluent.connect.jdbc.JdbcSourceConnector");

        // Connection
        config.put("connection.url",
"jdbc:postgresql://localhost:5432/analytics");
        config.put("connection.user", "analytics_user");
        config.put("connection.password", "password");

        // Custom query mode
        config.put("mode", "bulk");
        config.put("query",
            "SELECT u.id, u.email, COUNT(o.id) as order_count, " +
            "       SUM(o.total) as total_spent " +
            "FROM users u " +
            "LEFT JOIN orders o ON u.id = o.user_id " +
            "WHERE u.updated_at > ? " +
            "GROUP BY u.id, u.email");

        config.put("topic.prefix", "analytics-");
        config.put("poll.interval.ms", "300000"); // 5 minutes

        return config;
    }
}
```

JDBC Sink Connector - Kafka to Database

```
public class JDBCSinkConnectorExample {

    public static Map<String, String> createPostgresSinkConfig() {
        Map<String, String> config = new HashMap<>();

        // Connector identification
        config.put("name", "postgres-users-sink");
        config.put("connector.class",
"io.confluent.connect.jdbc.JdbcSinkConnector");

        // Database connection
        config.put("connection.url",
"jdbc:postgresql://localhost:5432/warehouse");
        config.put("connection.user", "warehouse_user");
        config.put("connection.password", "password");

        // Topics configuration
        config.put("topics", "postgres-users,postgres-orders");
        config.put("topics.regex", "postgres-.*");

        // Table configuration
        config.put("table.name.format", "kafka_${topic}");
        config.put("auto.create", "true");
        config.put("auto.evolve", "true");

        // Insert mode
        config.put("insert.mode", "insert");
        config.put("batch.size", "3000");
        config.put("max.retries", "10");
        config.put("retry.backoff.ms", "3000");

        // Field mapping
        config.put("fields.whitelist", "id,name,email,created_at");
        config.put("pk.mode", "record_key");
        config.put("pk.fields", "id");

        // Data type mapping
        config.put("db.timezone", "UTC");

        return config;
    }

    public static Map<String, String> createUpsertSinkConfig() {
        Map<String, String> config = new HashMap<>();

        config.put("name", "postgres-upsert-sink");
        config.put("connector.class",
"io.confluent.connect.jdbc.JdbcSinkConnector");
```

```
        // Connection
        config.put("connection.url",
"jdbc:postgresql://localhost:5432/warehouse");
        config.put("connection.user", "warehouse_user");
        config.put("connection.password", "password");

        // Topics
        config.put("topics", "user-updates");

        // Upsert configuration
        config.put("insert.mode", "upsert");
        config.put("pk.mode", "record_value");
        config.put("pk.fields", "id");

        // Table configuration
        config.put("table.name.format", "users");
        config.put("auto.create", "false"); // Table should exist
        config.put("auto.evolve", "true");

        // Batching
        config.put("batch.size", "1000");

        return config;
    }

    public static void demonstrateJDBCConnectors() {
        KafkaConnectRestClient client = new
KafkaConnectRestClient("http://localhost:8083");

        try {
            // Create source connector
            Map<String, String> sourceConfig = createIncrementalSourceConfig();
            client.createOrUpdateConnector("postgres-users-source", sourceConfig);

            // Create sink connector
            Map<String, String> sinkConfig = createPostgresSinkConfig();
            client.createOrUpdateConnector("postgres-users-sink", sinkConfig);

            // Monitor status
            Thread.sleep(5000);
            ConnectorStatus sourceStatus = client.getConnectorStatus("postgres-
users-source");
            ConnectorStatus sinkStatus = client.getConnectorStatus("postgres-
users-sink");

            System.out.println("Source connector: " + sourceStatus);
            System.out.println("Sink connector: " + sinkStatus);

        } catch (Exception e) {
            System.err.println("Error setting up JDBC connectors: " +
e.getMessage());
        }
```

```
    }
}
```

Debezium CDC

Debezium MySQL Connector

```
public class DebeziumMySQLConnectorExample {

    public static Map<String, String> createMySQLCDCConfig() {
        Map<String, String> config = new HashMap<>();

        // Connector identification
        config.put("name", "mysql-cdc-connector");
        config.put("connector.class",
"io.debezium.connector.mysql.MySqlConnector");

        // Database connection
        config.put("database.hostname", "localhost");
        config.put("database.port", "3306");
        config.put("database.user", "debezium");
        config.put("database.password", "password");
        config.put("database.server.id", "184054");
        config.put("database.server.name", "ecommerce-mysql");

        // Database selection
        config.put("database.include.list", "ecommerce");
        config.put("table.include.list",
"ecommerce.users,ecommerce.orders,ecommerce.products");

        // Binlog configuration
        config.put("database.history.kafka.bootstrap.servers", "localhost:9092");
        config.put("database.history.kafka.topic", "ecommerce-mysql-history");

        // Topic routing
        config.put("topic.prefix", "mysql-cdc");

        // Snapshot configuration
        config.put("snapshot.mode", "initial");
        config.put("snapshot.locking.mode", "minimal");
        config.put("snapshot.new.tables", "parallel");

        // Schema changes
        config.put("schema.history.internal.kafka.bootstrap.servers",
"localhost:9092");
        config.put("schema.history.internal.kafka.topic", "mysql-schema-changes");

        // Message key configuration
        config.put("message.key.columns",
"ecommerce.users:id;ecommerce.orders:id");
```

```
        // Transforms (see SMT section)
        config.put("transforms", "unwrap");
        config.put("transforms.unwrap.type",
"io.debezium.transforms.ExtractNewRecordState");

        return config;
    }

    public static Map<String, String> createPostgreSQLCDCConfig() {
        Map<String, String> config = new HashMap<>();

        // Connector identification
        config.put("name", "postgres-cdc-connector");
        config.put("connector.class",
"io.debezium.connector.postgresql.PostgresConnector");

        // Database connection
        config.put("database.hostname", "localhost");
        config.put("database.port", "5432");
        config.put("database.user", "postgres");
        config.put("database.password", "password");
        config.put("database.dbname", "ecommerce");
        config.put("database.server.name", "ecommerce-postgres");

        // WAL configuration
        config.put("slot.name", "debezium_slot");
        config.put("plugin.name", "pgoutput");

        // Schema and table filtering
        config.put("schema.include.list", "public");
        config.put("table.include.list",
"public.users,public.orders,public.order_items");

        // Topic configuration
        config.put("topic.prefix", "postgres-cdc");

        // Snapshot configuration
        config.put("snapshot.mode", "initial");

        // Publication configuration (PostgreSQL 10+)
        config.put("publication.name", "debezium_publication");
        config.put("publication.autocreate.mode", "filtered");

        return config;
    }

    // CDC Event Processing Example
    public static void processCDCEvents() {
        Properties props = new Properties();
        props.put("bootstrap.servers", "localhost:9092");
        props.put("group.id", "cdc-processor");
        props.put("key.deserializer",
"org.apache.kafka.common.serialization.StringDeserializer");
        props.put("value.deserializer",
```

```
"org.apache.kafka.common.serialization.StringDeserializer");
        props.put("auto.offset.reset", "earliest");

        try (KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props))
{
            consumer.subscribe(java.util.Arrays.asList(
                "mysql-cdc.ecommerce.users",
                "mysql-cdc.ecommerce.orders"
            ));

            while (true) {
                ConsumerRecords<String, String> records =
consumer.poll(Duration.ofMillis(1000));

                for (ConsumerRecord<String, String> record : records) {
                    processCDCRecord(record);
                }
            }
        } catch (Exception e) {
            System.err.println("Error processing CDC events: " + e.getMessage());
        }
    }

    private static void processCDCRecord(ConsumerRecord<String, String> record) {
        try {
            ObjectMapper mapper = new ObjectMapper();
            JsonNode event = mapper.readTree(record.value());

            String operation = event.has("op") ? event.get("op").asText() :
"unknown";
            String table = extractTableName(record.topic());

            System.out.printf("CDC Event: table=%s, operation=%s, key=%s%n",
                table, operation, record.key());

            switch (operation) {
                case "c": // Create
                    handleInsert(table, event.get("after"));
                    break;
                case "u": // Update
                    handleUpdate(table, event.get("before"), event.get("after"));
                    break;
                case "d": // Delete
                    handleDelete(table, event.get("before"));
                    break;
                case "r": // Read (snapshot)
                    handleSnapshot(table, event.get("after"));
                    break;
                default:
                    System.out.println("Unknown operation: " + operation);
            }

        } catch (Exception e) {
            System.err.println("Error processing CDC record: " + e.getMessage());
```

```
        }
    }

    private static String extractTableName(String topic) {
        // Topic format: mysql-cdc.database.table
        String[] parts = topic.split("\\.");
        return parts.length >= 3 ? parts[2] : "unknown";
    }

    private static void handleInsert(String table, JsonNode after) {
        System.out.println("INSERT into " + table + ": " + after);
        // Implement your insert logic
    }

    private static void handleUpdate(String table, JsonNode before, JsonNode
after) {
        System.out.println("UPDATE " + table + " from " + before + " to " +
after);
        // Implement your update logic
    }

    private static void handleDelete(String table, JsonNode before) {
        System.out.println("DELETE from " + table + ": " + before);
        // Implement your delete logic
    }

    private static void handleSnapshot(String table, JsonNode data) {
        System.out.println("SNAPSHOT " + table + ": " + data);
        // Implement your snapshot logic
    }
}
```

Elasticsearch/S3 Connectors

Elasticsearch Sink Connector

```
public class ElasticsearchConnectorExample {

    public static Map<String, String> createElasticsearchSinkConfig() {
        Map<String, String> config = new HashMap<>();

        // Connector identification
        config.put("name", "elasticsearch-sink-connector");
        config.put("connector.class",
"io.confluent.connect.elasticsearch.ElasticsearchSinkConnector");

        // Elasticsearch connection
        config.put("connection.url", "http://localhost:9200");
        config.put("connection.username", "elastic");
        config.put("connection.password", "changeme");
```

```
        // SSL configuration (if needed)
        config.put("connection.ssl.enabled", "false");
        config.put("connection.ssl.keystore.location", "/path/to/keystore.jks");
        config.put("connection.ssl.keystore.password", "keystore-password");
        config.put("connection.ssl.truststore.location",
"/path/to/truststore.jks");
        config.put("connection.ssl.truststore.password", "truststore-password");

        // Topics and routing
        config.put("topics", "mysql-cdc.ecommerce.users,mysql-
cdc.ecommerce.orders");
        config.put("topic.index.map",
            "mysql-cdc.ecommerce.users:users,mysql-cdc.ecommerce.orders:orders");

        // Document configuration
        config.put("type.name", "_doc");
        config.put("key.ignore", "false");
        config.put("schema.ignore", "true");

        // Batch configuration
        config.put("batch.size", "2000");
        config.put("max.in.flight.requests", "5");
        config.put("max.buffered.records", "20000");
        config.put("linger.ms", "1");
        config.put("flush.timeout.ms", "10000");

        // Error handling
        config.put("behavior.on.null.values", "ignore");
        config.put("behavior.on.malformed.documents", "warn");
        config.put("drop.invalid.message", "true");

        // Mapping configuration
        config.put("schema.ignore", "true");
        config.put("compact.map.entries", "true");

        return config;
    }

    public static Map<String, String> createAdvancedElasticsearchConfig() {
        Map<String, String> config = createElasticsearchSinkConfig();

        // Advanced routing based on record content
        config.put("topic.schema.ignore", "true");

        // Custom transformations
        config.put("transforms", "addTimestamp,extractKey");
        config.put("transforms.addTimestamp.type",
"org.apache.kafka.connect.transforms.InsertField$Value");
        config.put("transforms.addTimestamp.timestamp.field", "indexed_at");

        config.put("transforms.extractKey.type",
"org.apache.kafka.connect.transforms.ValueToKey");
        config.put("transforms.extractKey.fields", "id");
```

```
        // Dynamic index names based on timestamp
        config.put("topic.index.map",
            "user-events:user-events-{yyyy.MM.dd},order-events:order-events-
{yyyy.MM.dd}");

        // Retry configuration
        config.put("max.retries", "5");
        config.put("retry.backoff.ms", "5000");

        return config;
    }

    // Elasticsearch monitoring example
    public static void monitorElasticsearchConnector(String connectorName) {
        KafkaConnectRestClient client = new
KafkaConnectRestClient("http://localhost:8083");

        try {
            // Get connector status
            ConnectorStatus status = client.getConnectorStatus(connectorName);
            System.out.println("Elasticsearch connector status: " + status);

            // Check if connector is healthy
            if (!"RUNNING".equals(status.getState())) {
                System.err.println("Connector is not running: " +
status.getState());

                // Try restarting
                client.restartConnector(connectorName);
                System.out.println("Attempted to restart connector");
            }

        } catch (Exception e) {
            System.err.println("Error monitoring Elasticsearch connector: " +
e.getMessage());
        }
    }
}
```

S3 Sink Connector

```
public class S3ConnectorExample {

    public static Map<String, String> createS3SinkConfig() {
        Map<String, String> config = new HashMap<>();

        // Connector identification
        config.put("name", "s3-sink-connector");
        config.put("connector.class", "io.confluent.connect.s3.S3SinkConnector");

        // AWS configuration
```

```
        config.put("aws.access.key.id", "YOUR_ACCESS_KEY");
        config.put("aws.secret.access.key", "YOUR_SECRET_KEY");
        config.put("s3.region", "us-west-2");
        config.put("s3.bucket.name", "kafka-connect-s3");

        // Topics configuration
        config.put("topics", "mysql-cdc.ecommerce.users,mysql-
cdc.ecommerce.orders");
        config.put("topics.dir", "topics");

        // Partitioning
        config.put("s3.part.size", "5242880"); // 5MB
        config.put("flush.size", "10000");
        config.put("rotate.interval.ms", "3600000"); // 1 hour
        config.put("rotate.schedule.interval.ms", "3600000");

        // File format
        config.put("format.class",
"io.confluent.connect.s3.format.json.JsonFormat");
        config.put("partitioner.class",
"io.confluent.connect.storage.partitioner.TimeBasedPartitioner");
        config.put("partition.duration.ms", "3600000"); // 1 hour partitions
        config.put("path.format", "'year'=YYYY/'month'=MM/'day'=dd/'hour'=HH");
        config.put("locale", "US");
        config.put("timezone", "UTC");

        // Compression
        config.put("s3.compression.type", "gzip");

        // Schema registry (if using Avro)
        config.put("schema.registry.url", "http://localhost:8081");

        return config;
    }

    public static Map<String, String> createParquetS3SinkConfig() {
        Map<String, String> config = new HashMap<>();

        // Basic configuration
        config.put("name", "s3-parquet-sink");
        config.put("connector.class", "io.confluent.connect.s3.S3SinkConnector");

        // AWS configuration
        config.put("aws.access.key.id", "YOUR_ACCESS_KEY");
        config.put("aws.secret.access.key", "YOUR_SECRET_KEY");
        config.put("s3.region", "us-west-2");
        config.put("s3.bucket.name", "data-lake-parquet");

        // Topics
        config.put("topics", "user-analytics,order-analytics");

        // Parquet format
        config.put("format.class",
"io.confluent.connect.s3.format.parquet.ParquetFormat");
```

```
        config.put("schema.registry.url", "http://localhost:8081");
        config.put("schema.compatibility", "BACKWARD");

        // Partitioning for analytics
        config.put("partitioner.class",
"io.confluent.connect.storage.partitioner.FieldPartitioner");
        config.put("partition.field.name", "department,region");

        // Larger flush sizes for analytics workloads
        config.put("flush.size", "50000");
        config.put("rotate.interval.ms", "7200000"); // 2 hours

        // Compression
        config.put("s3.compression.type", "snappy");

        return config;
    }

    // S3 file organization example
    public static void demonstrateS3Organization() {
        /*
         * S3 Directory Structure Example:
         *
         * kafka-connect-s3/
         * ├── topics/
         * │   ├── mysql-cdc.ecommerce.users/
         * │   │   ├── year=2025/month=09/day=21/hour=10/
         * │   │   │   ├── mysql-cdc.ecommerce.users+0+0000000000.json.gz
         * │   │   │   └── mysql-cdc.ecommerce.users+0+0000010000.json.gz
         * │   │   └── year=2025/month=09/day=21/hour=11/
         * │   │       └── mysql-cdc.ecommerce.users+0+0000020000.json.gz
         * │   └── mysql-cdc.ecommerce.orders/
         * │       └── year=2025/month=09/day=21/hour=10/
         * │           └── mysql-cdc.ecommerce.orders+0+0000000000.json.gz
         *
         * File naming convention:
         * {topic}+{partition}+{start_offset}.{format}.{compression}
         */

        System.out.println("S3 connector will organize files by:");
        System.out.println("1. Topic name");
        System.out.println("2. Time-based partitioning (year/month/day/hour)");
        System.out.println("3. Partition and offset information");
        System.out.println("4. Configurable compression (gzip, snappy, etc.)");
    }
}
```

🛠️ Customization

Writing Custom Connectors

Custom Source Connector Example

```
import org.apache.kafka.connect.source.SourceConnector;
import org.apache.kafka.connect.source.SourceTask;
import org.apache.kafka.connect.connector.Task;
import org.apache.kafka.common.config.ConfigDef;
import org.apache.kafka.common.config.AbstractConfig;
import java.util.Map;
import java.util.List;
import java.util.ArrayList;

public class CustomFileSourceConnector extends SourceConnector {

    private Map<String, String> configProps;

    @Override
    public String version() {
        return "1.0.0";
    }

    @Override
    public void start(Map<String, String> props) {
        this.configProps = props;

        // Validate configuration
        CustomFileSourceConfig config = new CustomFileSourceConfig(props);

        // Additional startup logic
        System.out.println("Starting Custom File Source Connector");
        System.out.println("Monitoring directory: " + config.getDirectoryPath());
        System.out.println("Topic: " + config.getTopic());
    }

    @Override
    public Class<? extends Task> taskClass() {
        return CustomFileSourceTask.class;
    }

    @Override
    public List<Map<String, String>> taskConfigs(int maxTasks) {
        List<Map<String, String>> configs = new ArrayList<>();

        // For simplicity, create one task
        // In production, you might split work across multiple tasks
        configs.add(configProps);

        return configs;
    }

    @Override
    public void stop() {
        System.out.println("Stopping Custom File Source Connector");
```

```
        // Cleanup logic
    }

    @Override
    public ConfigDef config() {
        return CustomFileSourceConfig.CONFIG_DEF;
    }

    // Configuration class
    public static class CustomFileSourceConfig extends AbstractConfig {

        public static final String DIRECTORY_PATH_CONFIG = "directory.path";
        public static final String TOPIC_CONFIG = "topic";
        public static final String POLL_INTERVAL_MS_CONFIG = "poll.interval.ms";
        public static final String FILE_PATTERN_CONFIG = "file.pattern";

        public static final ConfigDef CONFIG_DEF = new ConfigDef()
            .define(DIRECTORY_PATH_CONFIG,
                ConfigDef.Type.STRING,
                ConfigDef.Importance.HIGH,
                "Directory path to monitor for files")
            .define(TOPIC_CONFIG,
                ConfigDef.Type.STRING,
                ConfigDef.Importance.HIGH,
                "Kafka topic to publish messages to")
            .define(POLL_INTERVAL_MS_CONFIG,
                ConfigDef.Type.LONG,
                5000L,
                ConfigDef.Importance.MEDIUM,
                "Polling interval in milliseconds")
            .define(FILE_PATTERN_CONFIG,
                ConfigDef.Type.STRING,
                ".*\\.txt$",
                ConfigDef.Importance.MEDIUM,
                "Regex pattern for files to process");

        public CustomFileSourceConfig(Map<String, String> props) {
            super(CONFIG_DEF, props);
        }

        public String getDirectoryPath() {
            return getString(DIRECTORY_PATH_CONFIG);
        }

        public String getTopic() {
            return getString(TOPIC_CONFIG);
        }

        public Long getPollInterval() {
            return getLong(POLL_INTERVAL_MS_CONFIG);
        }

        public String getFilePattern() {
            return getString(FILE_PATTERN_CONFIG);
```

```
        }
    }
}
```

Custom Source Task

```
import org.apache.kafka.connect.source.SourceTask;
import org.apache.kafka.connect.source.SourceRecord;
import org.apache.kafka.connect.data.Schema;
import org.apache.kafka.connect.data.SchemaBuilder;
import org.apache.kafka.connect.data.Struct;

import java.io.*;
import java.nio.file.*;
import java.util.*;
import java.util.concurrent.ConcurrentHashMap;
import java.util.regex.Pattern;

public class CustomFileSourceTask extends SourceTask {

    private CustomFileSourceConnector.CustomFileSourceConfig config;
    private Path directoryPath;
    private Pattern filePattern;
    private Map<String, Long> fileOffsets;
    private long lastPollTime;

    private static final String FILENAME_FIELD = "filename";
    private static final String CONTENT_FIELD = "content";
    private static final String LINE_NUMBER_FIELD = "line_number";
    private static final String TIMESTAMP_FIELD = "timestamp";

    private static final Schema VALUE_SCHEMA = SchemaBuilder.struct()
        .name("file.record")
        .field(FILENAME_FIELD, Schema.STRING_SCHEMA)
        .field(CONTENT_FIELD, Schema.STRING_SCHEMA)
        .field(LINE_NUMBER_FIELD, Schema.INT64_SCHEMA)
        .field(TIMESTAMP_FIELD, Schema.INT64_SCHEMA)
        .build();

    @Override
    public String version() {
        return "1.0.0";
    }

    @Override
    public void start(Map<String, String> props) {
        config = new CustomFileSourceConnector.CustomFileSourceConfig(props);
        directoryPath = Paths.get(config.getDirectoryPath());
        filePattern = Pattern.compile(config.getFilePattern());
        fileOffsets = new ConcurrentHashMap<>();
        lastPollTime = System.currentTimeMillis();
```

```
        // Load existing offsets from Kafka Connect offset storage
        loadOffsets();

        System.out.println("Started Custom File Source Task");
        System.out.println("Monitoring: " + directoryPath);
    }

    @Override
    public List<SourceRecord> poll() throws InterruptedException {
        List<SourceRecord> records = new ArrayList<>();

        try {
            // Sleep to avoid excessive polling
            Thread.sleep(config.getPollInterval());

            // Scan directory for files
            try (DirectoryStream<Path> stream =
Files.newDirectoryStream(directoryPath)) {
                for (Path filePath : stream) {
                    if (shouldProcessFile(filePath)) {
                        records.addAll(processFile(filePath));
                    }
                }
            }

        } catch (IOException e) {
            System.err.println("Error polling directory: " + e.getMessage());
        }

        return records;
    }

    private boolean shouldProcessFile(Path filePath) {
        if (Files.isDirectory(filePath)) {
            return false;
        }

        String filename = filePath.getFileName().toString();
        if (!filePattern.matcher(filename).matches()) {
            return false;
        }

        // Check if file has been modified since last poll
        try {
            long lastModified = Files.getLastModifiedTime(filePath).toMillis();
            return lastModified > lastPollTime;
        } catch (IOException e) {
            System.err.println("Error checking file modification time: " +
e.getMessage());
            return false;
        }
    }
```

```
    private List<SourceRecord> processFile(Path filePath) {
        List<SourceRecord> records = new ArrayList<>();
        String filename = filePath.getFileName().toString();

        try {
            Long currentOffset = fileOffsets.getOrDefault(filename, 0L);

            try (BufferedReader reader = Files.newBufferedReader(filePath)) {
                String line;
                long lineNumber = 0;

                // Skip lines we've already processed
                while (lineNumber < currentOffset && (line = reader.readLine()) !=
null) {
                    lineNumber++;
                }

                // Process new lines
                while ((line = reader.readLine()) != null) {
                    lineNumber++;

                    // Create source record
                    SourceRecord record = createSourceRecord(filename, line,
lineNumber);
                    records.add(record);

                    // Update offset
                    fileOffsets.put(filename, lineNumber);
                }
            }

        } catch (IOException e) {
            System.err.println("Error processing file " + filename + ": " +
e.getMessage());
        }

        return records;
    }

    private SourceRecord createSourceRecord(String filename, String content, long
lineNumber) {
        // Source partition - identifies the file
        Map<String, Object> sourcePartition = new HashMap<>();
        sourcePartition.put("filename", filename);

        // Source offset - line number in file
        Map<String, Object> sourceOffset = new HashMap<>();
        sourceOffset.put("position", lineNumber);

        // Record value
        Struct value = new Struct(VALUE_SCHEMA)
            .put(FILENAME_FIELD, filename)
            .put(CONTENT_FIELD, content)
            .put(LINE_NUMBER_FIELD, lineNumber)
```

```
                .put(TIMESTAMP_FIELD, System.currentTimeMillis());

            return new SourceRecord(
                sourcePartition,           // Source partition
                sourceOffset,              // Source offset
                config.getTopic(),         // Topic
                null,                      // Partition (let Kafka decide)
                Schema.STRING_SCHEMA,      // Key schema
                filename,                  // Key (filename)
                VALUE_SCHEMA,              // Value schema
                value                      // Value
            );
        }

        private void loadOffsets() {
            // In a real implementation, you would load offsets from Connect's offset
storage
            // For this example, we'll start from the beginning
            System.out.println("Loading offsets from Connect offset storage");
        }

        @Override
        public void stop() {
            System.out.println("Stopping Custom File Source Task");
            // Cleanup resources
        }
    }
```

Custom Sink Connector Example

```
    import org.apache.kafka.connect.sink.SinkConnector;
    import org.apache.kafka.connect.sink.SinkTask;
    import org.apache.kafka.connect.connector.Task;
    import org.apache.kafka.common.config.ConfigDef;
    import org.apache.kafka.common.config.AbstractConfig;

    public class CustomFileSinkConnector extends SinkConnector {

        private Map<String, String> configProps;

        @Override
        public String version() {
            return "1.0.0";
        }

        @Override
        public void start(Map<String, String> props) {
            this.configProps = props;

            CustomFileSinkConfig config = new CustomFileSinkConfig(props);
            System.out.println("Starting Custom File Sink Connector");
```

```
        System.out.println("Output directory: " + config.getOutputDirectory());
    }

    @Override
    public Class<? extends Task> taskClass() {
        return CustomFileSinkTask.class;
    }

    @Override
    public List<Map<String, String>> taskConfigs(int maxTasks) {
        List<Map<String, String>> configs = new ArrayList<>();

        // Create task configurations
        for (int i = 0; i < maxTasks; i++) {
            Map<String, String> taskConfig = new HashMap<>(configProps);
            taskConfig.put("task.id", String.valueOf(i));
            configs.add(taskConfig);
        }

        return configs;
    }

    @Override
    public void stop() {
        System.out.println("Stopping Custom File Sink Connector");
    }

    @Override
    public ConfigDef config() {
        return CustomFileSinkConfig.CONFIG_DEF;
    }

    // Configuration class
    public static class CustomFileSinkConfig extends AbstractConfig {

        public static final String OUTPUT_DIRECTORY_CONFIG = "output.directory";
        public static final String FILE_EXTENSION_CONFIG = "file.extension";
        public static final String FLUSH_SIZE_CONFIG = "flush.size";
        public static final String ROTATE_INTERVAL_MS_CONFIG =
"rotate.interval.ms";

        public static final ConfigDef CONFIG_DEF = new ConfigDef()
            .define(OUTPUT_DIRECTORY_CONFIG,
                ConfigDef.Type.STRING,
                ConfigDef.Importance.HIGH,
                "Directory to write output files")
            .define(FILE_EXTENSION_CONFIG,
                ConfigDef.Type.STRING,
                ".txt",
                ConfigDef.Importance.MEDIUM,
                "File extension for output files")
            .define(FLUSH_SIZE_CONFIG,
                ConfigDef.Type.INT,
                1000,
```

```
                ConfigDef.Importance.MEDIUM,
                "Number of records before flushing to file")
            .define(ROTATE_INTERVAL_MS_CONFIG,
                ConfigDef.Type.LONG,
                3600000L, // 1 hour
                ConfigDef.Importance.MEDIUM,
                "File rotation interval in milliseconds");

        public CustomFileSinkConfig(Map<String, String> props) {
            super(CONFIG_DEF, props);
        }

        public String getOutputDirectory() {
            return getString(OUTPUT_DIRECTORY_CONFIG);
        }

        public String getFileExtension() {
            return getString(FILE_EXTENSION_CONFIG);
        }

        public Integer getFlushSize() {
            return getInt(FLUSH_SIZE_CONFIG);
        }

        public Long getRotateInterval() {
            return getLong(ROTATE_INTERVAL_MS_CONFIG);
        }
    }
}
```

Custom Sink Task

```
import org.apache.kafka.connect.sink.SinkTask;
import org.apache.kafka.connect.sink.SinkRecord;
import java.io.*;
import java.nio.file.*;
import java.util.*;

public class CustomFileSinkTask extends SinkTask {

    private CustomFileSinkConnector.CustomFileSinkConfig config;
    private Path outputDirectory;
    private Map<String, BufferedWriter> writers;
    private Map<String, Integer> recordCounts;
    private Map<String, Long> lastRotateTime;

    @Override
    public String version() {
        return "1.0.0";
    }
```

```
    @Override
    public void start(Map<String, String> props) {
        config = new CustomFileSinkConnector.CustomFileSinkConfig(props);
        outputDirectory = Paths.get(config.getOutputDirectory());
        writers = new HashMap<>();
        recordCounts = new HashMap<>();
        lastRotateTime = new HashMap<>();

        // Create output directory if it doesn't exist
        try {
            Files.createDirectories(outputDirectory);
        } catch (IOException e) {
            throw new RuntimeException("Failed to create output directory", e);
        }

        System.out.println("Started Custom File Sink Task");
        System.out.println("Output directory: " + outputDirectory);
    }

    @Override
    public void put(Collection<SinkRecord> records) {
        for (SinkRecord record : records) {
            try {
                writeRecord(record);
            } catch (IOException e) {
                System.err.println("Error writing record: " + e.getMessage());
                // In production, you might want to implement retry logic
            }
        }

        // Check if any files need rotation
        checkFileRotation();
    }

    private void writeRecord(SinkRecord record) throws IOException {
        String topic = record.topic();
        String filename = generateFilename(topic);

        BufferedWriter writer = getWriter(topic, filename);

        // Write record to file
        String recordLine = formatRecord(record);
        writer.write(recordLine);
        writer.newLine();

        // Update record count
        recordCounts.put(topic, recordCounts.getOrDefault(topic, 0) + 1);

        // Check if we need to flush
        if (recordCounts.get(topic) >= config.getFlushSize()) {
            writer.flush();
            recordCounts.put(topic, 0);
        }
    }
```

```
    private BufferedWriter getWriter(String topic, String filename) throws
IOException {
        if (!writers.containsKey(topic)) {
            Path filePath = outputDirectory.resolve(filename);
            BufferedWriter writer = Files.newBufferedWriter(filePath,
                StandardOpenOption.CREATE, StandardOpenOption.APPEND);
            writers.put(topic, writer);
            lastRotateTime.put(topic, System.currentTimeMillis());
        }

        return writers.get(topic);
    }

    private String generateFilename(String topic) {
        long timestamp = System.currentTimeMillis();
        return String.format("%s-%d%s", topic, timestamp,
config.getFileExtension());
    }

    private String formatRecord(SinkRecord record) {
        // Format record as JSON or custom format
        return String.format("
{\"topic\":\"%s\",\"partition\":%d,\"offset\":%d,\"key\":\"%s\",\"value\":\"%s\",\
"timestamp\":%d}",
            record.topic(),
            record.kafkaPartition(),
            record.kafkaOffset(),
            record.key(),
            record.value(),
            record.timestamp());
    }

    private void checkFileRotation() {
        long currentTime = System.currentTimeMillis();

        for (Map.Entry<String, Long> entry : lastRotateTime.entrySet()) {
            String topic = entry.getKey();
            Long lastRotate = entry.getValue();

            if (currentTime - lastRotate > config.getRotateInterval()) {
                rotateFile(topic);
            }
        }
    }

    private void rotateFile(String topic) {
        try {
            BufferedWriter writer = writers.get(topic);
            if (writer != null) {
                writer.close();
                writers.remove(topic);
                lastRotateTime.put(topic, System.currentTimeMillis());
                recordCounts.put(topic, 0);
```

```
                System.out.println("Rotated file for topic: " + topic);
            }
        } catch (IOException e) {
            System.err.println("Error rotating file for topic " + topic + ": " +
e.getMessage());
        }
    }

    @Override
    public void flush(Map<org.apache.kafka.common.TopicPartition,
org.apache.kafka.clients.consumer.OffsetAndMetadata> offsets) {
        // Flush all writers
        for (BufferedWriter writer : writers.values()) {
            try {
                writer.flush();
            } catch (IOException e) {
                System.err.println("Error flushing writer: " + e.getMessage());
            }
        }
    }

    @Override
    public void stop() {
        // Close all writers
        for (BufferedWriter writer : writers.values()) {
            try {
                writer.close();
            } catch (IOException e) {
                System.err.println("Error closing writer: " + e.getMessage());
            }
        }

        writers.clear();
        recordCounts.clear();
        lastRotateTime.clear();

        System.out.println("Stopped Custom File Sink Task");
    }
}
```

Single Message Transforms (SMTs)

Custom SMT Example

```
import org.apache.kafka.common.config.ConfigDef;
import org.apache.kafka.connect.connector.ConnectRecord;
import org.apache.kafka.connect.data.Schema;
import org.apache.kafka.connect.data.SchemaBuilder;
import org.apache.kafka.connect.data.Struct;
import org.apache.kafka.connect.transforms.Transformation;
```

```
import org.apache.kafka.connect.transforms.util.SimpleConfig;

import java.util.Map;
import java.util.regex.Pattern;

public class MaskSensitiveData<R extends ConnectRecord<R>> implements
Transformation<R> {

    public static final String FIELD_NAME_CONFIG = "field.name";
    public static final String MASK_PATTERN_CONFIG = "mask.pattern";
    public static final String REPLACEMENT_CONFIG = "replacement";

    private static final ConfigDef CONFIG_DEF = new ConfigDef()
        .define(FIELD_NAME_CONFIG, ConfigDef.Type.STRING,
ConfigDef.Importance.HIGH,
            "Field name to mask")
        .define(MASK_PATTERN_CONFIG, ConfigDef.Type.STRING, "\\d{4}-\\d{4}-\\d{4}-
\\d{4}",
            ConfigDef.Importance.MEDIUM, "Regex pattern to match sensitive data")
        .define(REPLACEMENT_CONFIG, ConfigDef.Type.STRING, "****-****-****-****",
            ConfigDef.Importance.MEDIUM, "Replacement text for masked data");

    private String fieldName;
    private Pattern maskPattern;
    private String replacement;

    @Override
    public void configure(Map<String, ?> configs) {
        SimpleConfig config = new SimpleConfig(CONFIG_DEF, configs);
        fieldName = config.getString(FIELD_NAME_CONFIG);
        maskPattern = Pattern.compile(config.getString(MASK_PATTERN_CONFIG));
        replacement = config.getString(REPLACEMENT_CONFIG);
    }

    @Override
    public R apply(R record) {
        if (record.value() == null) {
            return record;
        }

        Object maskedValue = maskSensitiveData(record.value(),
record.valueSchema());

        return record.newRecord(
            record.topic(),
            record.kafkaPartition(),
            record.keySchema(),
            record.key(),
            record.valueSchema(),
            maskedValue,
            record.timestamp()
        );
    }
```

```
    private Object maskSensitiveData(Object value, Schema schema) {
        if (schema != null && schema.type() == Schema.Type.STRUCT) {
            return maskStructData((Struct) value, schema);
        } else if (value instanceof String) {
            return maskString((String) value);
        } else if (value instanceof Map) {
            return maskMapData((Map<String, Object>) value);
        }

        return value;
    }

    private Struct maskStructData(Struct struct, Schema schema) {
        Struct newStruct = new Struct(schema);

        for (org.apache.kafka.connect.data.Field field : schema.fields()) {
            Object fieldValue = struct.get(field);

            if (field.name().equals(fieldName) && fieldValue instanceof String) {
                String maskedValue = maskString((String) fieldValue);
                newStruct.put(field.name(), maskedValue);
            } else {
                newStruct.put(field.name(), fieldValue);
            }
        }

        return newStruct;
    }

    @SuppressWarnings("unchecked")
    private Map<String, Object> maskMapData(Map<String, Object> map) {
        Map<String, Object> newMap = new HashMap<>(map);

        if (newMap.containsKey(fieldName)) {
            Object value = newMap.get(fieldName);
            if (value instanceof String) {
                newMap.put(fieldName, maskString((String) value));
            }
        }

        return newMap;
    }

    private String maskString(String input) {
        return maskPattern.matcher(input).replaceAll(replacement);
    }

    @Override
    public ConfigDef config() {
        return CONFIG_DEF;
    }

    @Override
    public void close() {
```

```
        // Cleanup if needed
    }
}
```

Built-in SMTs Usage Examples

```
public class SMTExamples {

    // Example: Using SMTs in connector configuration
    public static Map<String, String> createConnectorWithSMTs() {
        Map<String, String> config = new HashMap<>();

        // Basic connector configuration
        config.put("name", "jdbc-source-with-transforms");
        config.put("connector.class",
"io.confluent.connect.jdbc.JdbcSourceConnector");
        config.put("connection.url",
"jdbc:postgresql://localhost:5432/ecommerce");
        config.put("table.whitelist", "users");
        config.put("topic.prefix", "postgres-");

        // Chain of Single Message Transforms
        config.put("transforms",
"addTimestamp,maskSSN,renameFields,filterFields");

        // 1. Add timestamp field
        config.put("transforms.addTimestamp.type",
            "org.apache.kafka.connect.transforms.InsertField$Value");
        config.put("transforms.addTimestamp.timestamp.field", "extracted_at");

        // 2. Mask sensitive data (custom SMT)
        config.put("transforms.maskSSN.type", "com.example.MaskSensitiveData");
        config.put("transforms.maskSSN.field.name", "ssn");
        config.put("transforms.maskSSN.mask.pattern", "\\d{3}-\\d{2}-\\d{4}");
        config.put("transforms.maskSSN.replacement", "***-**-****");

        // 3. Rename fields
        config.put("transforms.renameFields.type",
            "org.apache.kafka.connect.transforms.ReplaceField$Value");
        config.put("transforms.renameFields.renames",
"first_name:firstName,last_name:lastName");

        // 4. Filter out fields
        config.put("transforms.filterFields.type",
            "org.apache.kafka.connect.transforms.ReplaceField$Value");
        config.put("transforms.filterFields.blacklist", "internal_id,temp_field");

        return config;
    }

    // Example: Complex transformation pipeline
```

```
    public static Map<String, String> createComplexTransformPipeline() {
        Map<String, String> config = new HashMap<>();

        config.put("name", "complex-transform-pipeline");
        config.put("connector.class",
"io.debezium.connector.mysql.MySqlConnector");

        // Database configuration
        config.put("database.hostname", "localhost");
        config.put("database.server.name", "ecommerce");
        config.put("table.include.list", "ecommerce.orders");

        // Complex transformation pipeline
        config.put("transforms",
            "unwrapCDC,extractKey,addMetadata,routeByRegion,formatCurrency");

        // 1. Unwrap Debezium CDC format
        config.put("transforms.unwrapCDC.type",
            "io.debezium.transforms.ExtractNewRecordState");
        config.put("transforms.unwrapCDC.drop.tombstones", "false");
        config.put("transforms.unwrapCDC.delete.handling.mode", "rewrite");

        // 2. Extract key from value
        config.put("transforms.extractKey.type",
            "org.apache.kafka.connect.transforms.ValueToKey");
        config.put("transforms.extractKey.fields", "order_id");

        // 3. Add metadata
        config.put("transforms.addMetadata.type",
            "org.apache.kafka.connect.transforms.InsertField$Value");
        config.put("transforms.addMetadata.static.field", "source");
        config.put("transforms.addMetadata.static.value", "mysql-orders");

        // 4. Route by region using header
        config.put("transforms.routeByRegion.type",
            "org.apache.kafka.connect.transforms.RegexRouter");
        config.put("transforms.routeByRegion.regex", ".*");
        config.put("transforms.routeByRegion.replacement",
"orders-${header:region}");

        // 5. Format currency (custom SMT)
        config.put("transforms.formatCurrency.type",
"com.example.FormatCurrency");
        config.put("transforms.formatCurrency.field.name", "total_amount");
        config.put("transforms.formatCurrency.currency.code", "USD");

        return config;
    }

    // Example: Conditional transformations
    public static Map<String, String> createConditionalTransforms() {
        Map<String, String> config = new HashMap<>();

        config.put("name", "conditional-transforms");
```

```
        config.put("connector.class",
"io.confluent.connect.jdbc.JdbcSourceConnector");

        // Transformation chain with predicates
        config.put("transforms", "filterCustomers,addRegion");

        // 1. Filter only customers (with predicate)
        config.put("transforms.filterCustomers.type",
            "org.apache.kafka.connect.transforms.Filter");
        config.put("transforms.filterCustomers.predicate", "isCustomer");

        // 2. Add region field
        config.put("transforms.addRegion.type",
            "org.apache.kafka.connect.transforms.InsertField$Value");
        config.put("transforms.addRegion.static.field", "region");
        config.put("transforms.addRegion.static.value", "north-america");

        // Predicate configuration
        config.put("predicates", "isCustomer");
        config.put("predicates.isCustomer.type",
            "org.apache.kafka.connect.transforms.predicates.HasHeaderKey");
        config.put("predicates.isCustomer.name", "record_type");

        return config;
    }

    // Custom SMT for currency formatting
    public static class FormatCurrency<R extends ConnectRecord<R>> implements
Transformation<R> {

        private String fieldName;
        private String currencyCode;

        @Override
        public void configure(Map<String, ?> configs) {
            fieldName = (String) configs.get("field.name");
            currencyCode = (String) configs.get("currency.code");
        }

        @Override
        public R apply(R record) {
            if (record.value() instanceof Struct) {
                Struct struct = (Struct) record.value();
                Object amount = struct.get(fieldName);

                if (amount instanceof Number) {
                    String formattedAmount = String.format("%s %.2f",
                        currencyCode, ((Number) amount).doubleValue());

                    Struct newStruct = new Struct(struct.schema());
                    for (org.apache.kafka.connect.data.Field field :
struct.schema().fields()) {
                        if (field.name().equals(fieldName)) {
                            newStruct.put(field.name(), formattedAmount);
```

```
                        } else {
                            newStruct.put(field.name(), struct.get(field));
                        }
                    }

                    return record.newRecord(
                        record.topic(), record.kafkaPartition(),
                        record.keySchema(), record.key(),
                        record.valueSchema(), newStruct,
                        record.timestamp()
                    );
                }
            }

            return record;
        }

        @Override
        public ConfigDef config() {
            return new ConfigDef()
                .define("field.name", ConfigDef.Type.STRING,
ConfigDef.Importance.HIGH, "Field to format")
                .define("currency.code", ConfigDef.Type.STRING, "USD",
ConfigDef.Importance.MEDIUM, "Currency code");
        }

        @Override
        public void close() {}
    }
}
```

Comprehensive Java Examples

Production Kafka Connect Management System

```
import java.util.concurrent.*;
import java.util.concurrent.atomic.AtomicBoolean;

/**
 * Production-ready Kafka Connect Management System
 * Handles connector lifecycle, monitoring, and health checks
 */
public class KafkaConnectManager {

    private final KafkaConnectRestClient restClient;
    private final ScheduledExecutorService scheduler;
    private final AtomicBoolean running;
    private final Map<String, ConnectorHealth> connectorHealth;

    public KafkaConnectManager(String connectUrl) {
```

```
        this.restClient = new KafkaConnectRestClient(connectUrl);
        this.scheduler = Executors.newScheduledThreadPool(3);
        this.running = new AtomicBoolean(false);
        this.connectorHealth = new ConcurrentHashMap<>();
    }

    public void start() {
        if (running.compareAndSet(false, true)) {
            // Start health monitoring
            scheduler.scheduleAtFixedRate(this::monitorConnectorHealth,
                30, 30, TimeUnit.SECONDS);

            // Start metrics collection
            scheduler.scheduleAtFixedRate(this::collectMetrics,
                60, 60, TimeUnit.SECONDS);

            // Start configuration backup
            scheduler.scheduleAtFixedRate(this::backupConfigurations,
                0, 1, TimeUnit.HOURS);

            System.out.println("Kafka Connect Manager started");
        }
    }

    public void stop() {
        if (running.compareAndSet(true, false)) {
            scheduler.shutdown();
            try {
                if (!scheduler.awaitTermination(30, TimeUnit.SECONDS)) {
                    scheduler.shutdownNow();
                }
            } catch (InterruptedException e) {
                scheduler.shutdownNow();
                Thread.currentThread().interrupt();
            }
            System.out.println("Kafka Connect Manager stopped");
        }
    }

    // Deploy multiple connectors with dependency management
    public void deployConnectorPipeline(List<ConnectorDeployment> deployments) {
        System.out.println("Deploying connector pipeline...");

        // Sort by dependency order
        deployments.sort((a, b) -> a.getOrder() - b.getOrder());

        for (ConnectorDeployment deployment : deployments) {
            try {
                deployConnectorWithRetry(deployment);

                // Wait for connector to be ready before deploying next
                waitForConnectorReady(deployment.getName(),
Duration.ofMinutes(5));
```

```
            } catch (Exception e) {
                System.err.printf("Failed to deploy connector %s: %s%n",
                    deployment.getName(), e.getMessage());

                // Decide whether to continue or stop pipeline deployment
                if (deployment.isCritical()) {
                    throw new RuntimeException("Critical connector deployment
failed", e);
                }
            }
        }

        System.out.println("Connector pipeline deployment completed");
    }

    private void deployConnectorWithRetry(ConnectorDeployment deployment)
            throws Exception {

        int maxRetries = 3;
        int attempt = 0;

        while (attempt < maxRetries) {
            try {
                restClient.createOrUpdateConnector(deployment.getName(),
                    deployment.getConfig());

                System.out.printf("Successfully deployed connector: %s%n",
                    deployment.getName());
                return;

            } catch (Exception e) {
                attempt++;
                if (attempt >= maxRetries) {
                    throw e;
                }

                System.err.printf("Deployment attempt %d failed for %s: %s%n",
                    attempt, deployment.getName(), e.getMessage());

                // Exponential backoff
                Thread.sleep(1000 * (1L << attempt));
            }
        }
    }

    private void waitForConnectorReady(String connectorName, Duration timeout)
            throws Exception {

        long endTime = System.currentTimeMillis() + timeout.toMillis();

        while (System.currentTimeMillis() < endTime) {
            try {
                ConnectorStatus status =
restClient.getConnectorStatus(connectorName);
```

```
                if ("RUNNING".equals(status.getState()) && status.getTaskCount() >
0) {
                    System.out.printf("Connector %s is ready%n", connectorName);
                    return;
                }

                Thread.sleep(5000);

            } catch (Exception e) {
                System.err.printf("Error checking connector %s status: %s%n",
                    connectorName, e.getMessage());
                Thread.sleep(5000);
            }
        }

        throw new TimeoutException("Connector " + connectorName +
            " did not become ready within " + timeout);
    }

    private void monitorConnectorHealth() {
        try {
            String[] connectors = restClient.listConnectors();

            for (String connectorName : connectors) {
                checkConnectorHealth(connectorName);
            }

        } catch (Exception e) {
            System.err.println("Error during health monitoring: " +
e.getMessage());
        }
    }

    private void checkConnectorHealth(String connectorName) {
        try {
            ConnectorStatus status = restClient.getConnectorStatus(connectorName);
            ConnectorHealth health =
connectorHealth.computeIfAbsent(connectorName,
                k -> new ConnectorHealth());

            health.updateStatus(status.getState(), status.getTaskCount());

            // Check for issues
            if (!"RUNNING".equals(status.getState())) {
                health.recordFailure();
                handleUnhealthyConnector(connectorName, status);
            } else {
                health.recordSuccess();
            }

            // Log health summary
            if (health.getConsecutiveFailures() > 0) {
                System.err.printf("Connector %s health: %d consecutive
```

```
failures%n",
                        connectorName, health.getConsecutiveFailures());
                }

            } catch (Exception e) {
                System.err.printf("Error checking health for connector %s: %s%n",
                    connectorName, e.getMessage());
            }
        }

        private void handleUnhealthyConnector(String connectorName, ConnectorStatus
status) {
            ConnectorHealth health = connectorHealth.get(connectorName);

            // Auto-restart after 3 consecutive failures
            if (health.getConsecutiveFailures() >= 3) {
                try {
                    System.out.printf("Auto-restarting unhealthy connector: %s%n",
connectorName);
                    restClient.restartConnector(connectorName);
                    health.recordRestart();

                } catch (Exception e) {
                    System.err.printf("Failed to restart connector %s: %s%n",
                        connectorName, e.getMessage());
                }
            }

            // Alert after 5 consecutive failures
            if (health.getConsecutiveFailures() >= 5) {
                sendAlert(String.format("Connector %s has failed %d times
consecutively",
                    connectorName, health.getConsecutiveFailures()));
            }
        }

        private void collectMetrics() {
            try {
                String[] connectors = restClient.listConnectors();

                System.out.println("\n=== Connector Metrics Summary ===");
                System.out.printf("Total Connectors: %d%n", connectors.length);

                int runningCount = 0;
                int failedCount = 0;

                for (String connectorName : connectors) {
                    try {
                        ConnectorStatus status =
restClient.getConnectorStatus(connectorName);

                        if ("RUNNING".equals(status.getState())) {
                            runningCount++;
                        } else {
```

```
                        failedCount++;
                    }

                    System.out.printf("  %s: %s (%d tasks)%n",
                        connectorName, status.getState(), status.getTaskCount());

                } catch (Exception e) {
                    failedCount++;
                    System.err.printf("  %s: ERROR - %s%n", connectorName,
e.getMessage());
                }
            }

            System.out.printf("Running: %d, Failed: %d%n", runningCount,
failedCount);

        } catch (Exception e) {
            System.err.println("Error collecting metrics: " + e.getMessage());
        }
    }

    private void backupConfigurations() {
        try {
            String[] connectors = restClient.listConnectors();

            System.out.println("Backing up connector configurations...");

            for (String connectorName : connectors) {
                try {
                    Map<String, String> config =
restClient.getConnectorConfig(connectorName);
                    saveConfigurationBackup(connectorName, config);

                } catch (Exception e) {
                    System.err.printf("Failed to backup config for %s: %s%n",
                        connectorName, e.getMessage());
                }
            }

        } catch (Exception e) {
            System.err.println("Error during configuration backup: " +
e.getMessage());
        }
    }

    private void saveConfigurationBackup(String connectorName, Map<String, String>
config) {
        // In production, save to file system, database, or configuration
management system
        System.out.printf("Backed up configuration for %s (%d properties)%n",
            connectorName, config.size());
    }

    private void sendAlert(String message) {
```

```
        // In production, integrate with alerting system (Slack, PagerDuty, etc.)
        System.err.println("ALERT: " + message);
    }

    // Helper classes
    public static class ConnectorDeployment {
        private final String name;
        private final Map<String, String> config;
        private final int order;
        private final boolean critical;

        public ConnectorDeployment(String name, Map<String, String> config,
                                   int order, boolean critical) {
            this.name = name;
            this.config = config;
            this.order = order;
            this.critical = critical;
        }

        // Getters
        public String getName() { return name; }
        public Map<String, String> getConfig() { return config; }
        public int getOrder() { return order; }
        public boolean isCritical() { return critical; }
    }

    public static class ConnectorHealth {
        private String lastState = "UNKNOWN";
        private int taskCount = 0;
        private int consecutiveFailures = 0;
        private int consecutiveSuccesses = 0;
        private long lastUpdate = System.currentTimeMillis();
        private int restartCount = 0;

        public void updateStatus(String state, int tasks) {
            this.lastState = state;
            this.taskCount = tasks;
            this.lastUpdate = System.currentTimeMillis();
        }

        public void recordFailure() {
            consecutiveFailures++;
            consecutiveSuccesses = 0;
        }

        public void recordSuccess() {
            consecutiveSuccesses++;
            consecutiveFailures = 0;
        }

        public void recordRestart() {
            restartCount++;
        }
```

```
        // Getters
        public String getLastState() { return lastState; }
        public int getTaskCount() { return taskCount; }
        public int getConsecutiveFailures() { return consecutiveFailures; }
        public int getConsecutiveSuccesses() { return consecutiveSuccesses; }
        public long getLastUpdate() { return lastUpdate; }
        public int getRestartCount() { return restartCount; }
    }

    // Usage example
    public static void main(String[] args) {
        KafkaConnectManager manager = new
KafkaConnectManager("http://localhost:8083");

        try {
            manager.start();

            // Create deployment pipeline
            List<ConnectorDeployment> pipeline = Arrays.asList(
                new ConnectorDeployment("postgres-source",
                    JDBCSourceConnectorExample.createIncrementalSourceConfig(), 1,
true),
                new ConnectorDeployment("elasticsearch-sink",
                    ElasticsearchConnectorExample.createElasticsearchSinkConfig(),
2, false),
                new ConnectorDeployment("s3-backup",
                    S3ConnectorExample.createS3SinkConfig(), 3, false)
            );

            // Deploy pipeline
            manager.deployConnectorPipeline(pipeline);

            // Keep running
            Thread.sleep(Long.MAX_VALUE);

        } catch (Exception e) {
            System.err.println("Manager failed: " + e.getMessage());
        } finally {
            manager.stop();
        }
    }
}
```

⚖️ Comparisons & Trade-offs

Deployment Mode Comparison

Feature	Standalone Mode	Distributed Mode
Setup Complexity	Simple	Complex

Feature	Standalone Mode	Distributed Mode
Fault Tolerance	None	High
Scalability	Limited	Horizontal
Configuration	File-based	REST API + Kafka topics
Use Case	Development, Testing	Production
Resource Usage	Low	Higher
Management	Manual	Automated

Connector Types Comparison

Aspect	Source Connectors	Sink Connectors
Data Flow	External System → Kafka	Kafka → External System
Offset Management	Track external system position	Track Kafka offsets
Error Handling	Retry reading from source	DLQ, ignore, or fail
Backpressure	Kafka topic capacity	External system capacity
Exactly-Once	Depends on source capabilities	Idempotent writes needed

Common Connectors Comparison

Connector	Type	Use Case	Complexity	Performance
JDBC	Source/Sink	Database integration	Medium	Good
Debezium	Source	Change Data Capture	High	Excellent
Elasticsearch	Sink	Search indexing	Low	Good
S3	Sink	Data lake/backup	Low	Excellent
File	Source/Sink	File system integration	Low	Good

Common Pitfalls & Best Practices

1. Configuration Issues

❌ Incorrect Offset Storage Configuration

```
// DON'T - Wrong offset storage for distributed mode
Map<String, String> badConfig = new HashMap<>();
badConfig.put("offset.storage.file.filename", "/tmp/connect.offsets"); // File
storage in distributed mode!
badConfig.put("group.id", "connect-cluster");
```

```
// DO - Proper distributed mode configuration
Map<String, String> goodConfig = new HashMap<>();
goodConfig.put("group.id", "connect-cluster");
goodConfig.put("offset.storage.topic", "connect-offsets");
goodConfig.put("offset.storage.replication.factor", "3");
goodConfig.put("offset.storage.partitions", "25");
```

❌ Missing Replication Factor

```
// DON'T - Default replication factor (might be 1)
config.put("config.storage.topic", "connect-configs");
```

```
// DO - Explicit replication factor for production
config.put("config.storage.topic", "connect-configs");
config.put("config.storage.replication.factor", "3");
```

2. Resource Management Issues

❌ No Resource Limits

```
// DON'T - Unbounded resource usage
Map<String, String> connectorConfig = new HashMap<>();
connectorConfig.put("tasks.max", "100"); // Too many tasks
connectorConfig.put("batch.size", "100000"); // Too large batches
```

```
// DO - Reasonable resource limits
Map<String, String> connectorConfig = new HashMap<>();
connectorConfig.put("tasks.max", "4"); // Based on partition count
connectorConfig.put("batch.size", "2000"); // Reasonable batch size
connectorConfig.put("poll.interval.ms", "5000"); // Don't poll too frequently
```

3. Error Handling Problems

❌ No Error Handling Strategy

```
// DON'T - Default error handling (might stop connector)
Map<String, String> config = new HashMap<>();
config.put("connector.class",
"io.confluent.connect.elasticsearch.ElasticsearchSinkConnector");
// No error handling configuration
```

```
// DO - Comprehensive error handling
Map<String, String> config = new HashMap<>();
config.put("connector.class",
"io.confluent.connect.elasticsearch.ElasticsearchSinkConnector");

// Error tolerance
config.put("errors.tolerance", "all");
config.put("errors.log.enable", "true");
config.put("errors.log.include.messages", "true");

// Dead letter queue
config.put("errors.deadletterqueue.topic.name", "connect-dlq");
config.put("errors.deadletterqueue.topic.replication.factor", "3");
config.put("errors.deadletterqueue.context.headers.enable", "true");

// Retry configuration
config.put("errors.retry.timeout", "300000"); // 5 minutes
config.put("errors.retry.delay.max.ms", "60000"); // 1 minute max delay
```

4. Security Oversights

❌ No Security Configuration

```
// DON'T - Plain text passwords and no SSL
config.put("connection.password", "plaintext-password");
config.put("connection.url", "jdbc:postgresql://localhost:5432/db");
```

```
// DO - Proper security configuration
// Use ConfigProvider for sensitive data
config.put("config.providers", "file");
config.put("config.providers.file.class",
"org.apache.kafka.common.config.provider.FileConfigProvider");

// Reference encrypted passwords
config.put("connection.password", "${file:/etc/kafka-
connect/secrets.properties:db.password}");

// SSL configuration
config.put("connection.url", "jdbc:postgresql://localhost:5432/db?
ssl=true&sslfactory=org.postgresql.ssl.NonValidatingFactory");

// Kafka security
config.put("bootstrap.servers", "localhost:9093");
config.put("security.protocol", "SSL");
config.put("ssl.keystore.location", "/etc/kafka-connect/keystore.jks");
```

```
config.put("ssl.keystore.password", "${file:/etc/kafka-
connect/secrets.properties:keystore.password}");
```

Best Practices Summary

☑ Configuration Best Practices

1. **Use distributed mode for production** - Better fault tolerance and scalability
2. **Set appropriate replication factors** - Ensure high availability for internal topics
3. **Configure error handling** - Use DLQ and error tolerance for resilience
4. **Secure sensitive data** - Use ConfigProvider for passwords and keys
5. **Monitor resource usage** - Set reasonable limits on tasks, batch sizes, and polling

☑ Operational Best Practices

1. **Monitor connector health** - Implement automated health checks and alerting
2. **Backup configurations** - Store connector configs in version control
3. **Use proper naming conventions** - Include environment and purpose in names
4. **Implement gradual rollouts** - Deploy connectors incrementally
5. **Plan for disaster recovery** - Document recovery procedures

☑ Development Best Practices

1. **Test with realistic data volumes** - Performance characteristics change with scale
2. **Implement custom SMTs carefully** - Test thoroughly and handle edge cases
3. **Use schema registry** - For better data governance and evolution
4. **Monitor offset lag** - Ensure connectors keep up with data flow
5. **Document dependencies** - Clearly specify connector deployment order

---

🌍 Real-World Use Cases

1. E-commerce Data Pipeline

```
public class EcommerceDataPipeline {

    public static List<ConnectorDeployment> createEcommercePipeline() {
        List<ConnectorDeployment> pipeline = new ArrayList<>();

        // 1. Source: MySQL database (orders, customers, products)
        Map<String, String> mysqlSource = new HashMap<>();
        mysqlSource.put("name", "mysql-ecommerce-source");
        mysqlSource.put("connector.class",
"io.debezium.connector.mysql.MySqlConnector");
        mysqlSource.put("database.hostname", "mysql.internal");
        mysqlSource.put("database.server.name", "ecommerce");
        mysqlSource.put("table.include.list",
"ecommerce.orders,ecommerce.customers,ecommerce.products");
```

```
        mysqlSource.put("transforms", "unwrap");
        mysqlSource.put("transforms.unwrap.type",
"io.debezium.transforms.ExtractNewRecordState");

        pipeline.add(new ConnectorDeployment("mysql-source", mysqlSource, 1,
true));

        // 2. Sink: Elasticsearch for search and analytics
        Map<String, String> esSink = new HashMap<>();
        esSink.put("name", "elasticsearch-ecommerce-sink");
        esSink.put("connector.class",
"io.confluent.connect.elasticsearch.ElasticsearchSinkConnector");
        esSink.put("topics",
"ecommerce.orders,ecommerce.customers,ecommerce.products");
        esSink.put("connection.url", "http://elasticsearch.internal:9200");
        esSink.put("type.name", "_doc");
        esSink.put("topic.index.map",
"ecommerce.orders:orders,ecommerce.customers:customers,ecommerce.products:products
");

        pipeline.add(new ConnectorDeployment("elasticsearch-sink", esSink, 2,
false));

        // 3. Sink: S3 for data lake
        Map<String, String> s3Sink = new HashMap<>();
        s3Sink.put("name", "s3-datalake-sink");
        s3Sink.put("connector.class", "io.confluent.connect.s3.S3SinkConnector");
        s3Sink.put("topics",
"ecommerce.orders,ecommerce.customers,ecommerce.products");
        s3Sink.put("s3.bucket.name", "ecommerce-datalake");
        s3Sink.put("format.class",
"io.confluent.connect.s3.format.parquet.ParquetFormat");
        s3Sink.put("partitioner.class",
"io.confluent.connect.storage.partitioner.TimeBasedPartitioner");
        s3Sink.put("path.format", "'year'=YYYY/'month'=MM/'day'=dd");
        s3Sink.put("rotate.interval.ms", "3600000"); // Hourly rotation

        pipeline.add(new ConnectorDeployment("s3-datalake-sink", s3Sink, 3,
false));

        // 4. Sink: PostgreSQL data warehouse
        Map<String, String> postgresWarehouse = new HashMap<>();
        postgresWarehouse.put("name", "postgres-warehouse-sink");
        postgresWarehouse.put("connector.class",
"io.confluent.connect.jdbc.JdbcSinkConnector");
        postgresWarehouse.put("topics", "ecommerce.orders,ecommerce.customers");
        postgresWarehouse.put("connection.url",
"jdbc:postgresql://warehouse.internal:5432/analytics");
        postgresWarehouse.put("auto.create", "true");
        postgresWarehouse.put("auto.evolve", "true");
        postgresWarehouse.put("insert.mode", "upsert");
        postgresWarehouse.put("pk.mode", "record_value");
        postgresWarehouse.put("pk.fields", "id");
```

```
        pipeline.add(new ConnectorDeployment("postgres-warehouse-sink",
postgresWarehouse, 4, false));

        return pipeline;
    }
}
```

2. IoT Sensor Data Processing

```
public class IoTSensorPipeline {

    public static List<ConnectorDeployment> createIoTPipeline() {
        List<ConnectorDeployment> pipeline = new ArrayList<>();

        // 1. Source: File-based sensor data ingestion
        Map<String, String> fileSource = new HashMap<>();
        fileSource.put("name", "iot-file-source");
        fileSource.put("connector.class", "com.custom.IoTFileSourceConnector");
        fileSource.put("directory.path", "/data/sensors");
        fileSource.put("file.pattern", ".*\\.sensor$");
        fileSource.put("topic", "raw-sensor-data");
        fileSource.put("poll.interval.ms", "1000");

        // Transform sensor data
        fileSource.put("transforms", "parseJson,addTimestamp,routeByType");
        fileSource.put("transforms.parseJson.type", "com.custom.ParseIoTData");
        fileSource.put("transforms.addTimestamp.type",
"org.apache.kafka.connect.transforms.InsertField$Value");
        fileSource.put("transforms.addTimestamp.timestamp.field", "processed_at");
        fileSource.put("transforms.routeByType.type",
"org.apache.kafka.connect.transforms.RegexRouter");
        fileSource.put("transforms.routeByType.regex", "raw-sensor-data");
        fileSource.put("transforms.routeByType.replacement",
"sensors-${header:sensor_type}");

        pipeline.add(new ConnectorDeployment("iot-file-source", fileSource, 1,
true));

        // 2. Sink: InfluxDB for time series data
        Map<String, String> influxSink = new HashMap<>();
        influxSink.put("name", "influxdb-timeseries-sink");
        influxSink.put("connector.class",
"com.influxdb.kafka.InfluxDBSinkConnector");
        influxSink.put("topics.regex", "sensors-.*");
        influxSink.put("influxdb.url", "http://influxdb.internal:8086");
        influxSink.put("influxdb.db", "iot_sensors");
        influxSink.put("measurement.name.format", "${topic}");

        pipeline.add(new ConnectorDeployment("influxdb-sink", influxSink, 2,
false));
```

```
        // 3. Sink: Alert system for anomalies
        Map<String, String> alertSink = new HashMap<>();
        alertSink.put("name", "alert-webhook-sink");
        alertSink.put("connector.class", "com.custom.WebhookSinkConnector");
        alertSink.put("topics", "sensor-alerts");
        alertSink.put("webhook.url", "http://alert-service.internal/webhook");
        alertSink.put("http.headers", "Content-
Type:application/json,Authorization:Bearer ${file:/secrets:webhook.token}");

        pipeline.add(new ConnectorDeployment("alert-webhook-sink", alertSink, 3,
true));

        return pipeline;
    }
}
```

3. Financial Data Compliance Pipeline

```
public class FinancialCompliancePipeline {

    public static List<ConnectorDeployment> createCompliancePipeline() {
        List<ConnectorDeployment> pipeline = new ArrayList<>();

        // 1. Source: Core banking system
        Map<String, String> bankingSource = new HashMap<>();
        bankingSource.put("name", "core-banking-source");
        bankingSource.put("connector.class",
"io.confluent.connect.jdbc.JdbcSourceConnector");
        bankingSource.put("connection.url", "${file:/secrets:banking.url}");
        bankingSource.put("connection.user", "${file:/secrets:banking.user}");
        bankingSource.put("connection.password",
"${file:/secrets:banking.password}");
        bankingSource.put("table.whitelist", "transactions,accounts,customers");
        bankingSource.put("mode", "timestamp+incrementing");
        bankingSource.put("timestamp.column.name", "updated_at");
        bankingSource.put("incrementing.column.name", "id");
        bankingSource.put("poll.interval.ms", "30000"); // 30 seconds for near
real-time

        // Privacy transformations
        bankingSource.put("transforms",
"maskPII,encryptSensitive,addAuditFields");
        bankingSource.put("transforms.maskPII.type", "com.bank.MaskPersonalData");
        bankingSource.put("transforms.maskPII.fields",
"ssn,account_number,routing_number");
        bankingSource.put("transforms.encryptSensitive.type",
"com.bank.EncryptField");
        bankingSource.put("transforms.encryptSensitive.fields",
"customer_id,transaction_amount");
        bankingSource.put("transforms.addAuditFields.type",
"org.apache.kafka.connect.transforms.InsertField$Value");
```

```
        bankingSource.put("transforms.addAuditFields.static.field",
"audit_source");
        bankingSource.put("transforms.addAuditFields.static.value", "core-
banking");

        pipeline.add(new ConnectorDeployment("banking-source", bankingSource, 1,
true));

        // 2. Sink: Compliance data warehouse
        Map<String, String> complianceSink = new HashMap<>();
        complianceSink.put("name", "compliance-warehouse-sink");
        complianceSink.put("connector.class",
"io.confluent.connect.jdbc.JdbcSinkConnector");
        complianceSink.put("topics", "banking-transactions,banking-
accounts,banking-customers");
        complianceSink.put("connection.url", "${file:/secrets:warehouse.url}");
        complianceSink.put("connection.user", "${file:/secrets:warehouse.user}");
        complianceSink.put("connection.password",
"${file:/secrets:warehouse.password}");
        complianceSink.put("table.name.format", "compliance_${topic}");
        complianceSink.put("auto.create", "true");
        complianceSink.put("auto.evolve", "false"); // Strict schema control
        complianceSink.put("insert.mode", "insert"); // Append-only for audit
trail

        // Enhanced error handling for compliance
        complianceSink.put("errors.tolerance", "none"); // No error tolerance for
compliance
        complianceSink.put("errors.log.enable", "true");
        complianceSink.put("errors.log.include.messages", "true");

        pipeline.add(new ConnectorDeployment("compliance-warehouse-sink",
complianceSink, 2, true));

        // 3. Sink: Encrypted backup to S3
        Map<String, String> backupSink = new HashMap<>();
        backupSink.put("name", "encrypted-backup-sink");
        backupSink.put("connector.class",
"io.confluent.connect.s3.S3SinkConnector");
        backupSink.put("topics", "banking-transactions,banking-accounts,banking-
customers");
        backupSink.put("s3.bucket.name", "financial-compliance-backup");
        backupSink.put("format.class",
"io.confluent.connect.s3.format.json.JsonFormat");

        // Encryption and retention
        backupSink.put("s3.ssea.name", "AES256");
        backupSink.put("s3.canned.acl", "private");
        backupSink.put("rotate.interval.ms", "86400000"); // Daily rotation

        // Compliance partitioning
        backupSink.put("partitioner.class",
"io.confluent.connect.storage.partitioner.TimeBasedPartitioner");
        backupSink.put("path.format", "'year'=YYYY/'month'=MM/'day'=dd");
```

```
        backupSink.put("timezone", "UTC");

        pipeline.add(new ConnectorDeployment("encrypted-backup-sink", backupSink,
3, true));

        return pipeline;
    }
}
```

Version Highlights

Kafka Connect Evolution Timeline

Version	Release Date	Major Features
4.0	September 2025	Enhanced REST API, improved error handling
3.0	September 2021	Better connector management, KRaft support
2.8	April 2021	Incremental cooperative rebalancing
2.4	December 2019	Enhanced SMT support, better monitoring
2.0	July 2018	Header support, improved error handling
1.0	October 2017	Schema evolution support
0.11	June 2017	Exactly-once semantics support
0.10	May 2016	**Initial Kafka Connect release**

Key Features by Version

Kafka Connect 4.0 (September 2025)

- ✨ **Enhanced REST API**: Better error responses and bulk operations
- ✨ **Improved Error Handling**: More granular error tolerance configuration
- ✨ **Better Monitoring**: Enhanced JMX metrics and health checks
- ✨ **Security Improvements**: Better integration with OAuth and RBAC

Kafka Connect 3.x Series

- **3.6** (October 2023): Enhanced connector plugin management
- **3.5** (June 2023): Improved offset management and recovery
- **3.4** (February 2023): Better resource management and scaling
- **3.3** (October 2022): Enhanced transform pipeline performance
- **3.2** (May 2022): Improved configuration validation
- **3.1** (January 2022): Better dead letter queue handling
- **3.0** (September 2021): **Major stability improvements**

Kafka Connect 2.x Highlights

- **2.8** (April 2021): Incremental cooperative rebalancing for Connect
- **2.7** (December 2020): Enhanced connector restart capabilities
- **2.6** (August 2020): Better offset commit strategies
- **2.5** (April 2020): Improved exactly-once support
- **2.4** (December 2019): **Enhanced SMT framework**

Notable Connector Ecosystem Growth

- **Debezium** (2016-present): Leading CDC connector platform
- **Confluent Hub** (2018-present): Connector distribution platform
- **Community Connectors** (2017-present): 100+ community-developed connectors

Current Recommendations (2025)

```
// Modern Kafka Connect configuration (4.0+)
public static Properties modernConnectConfig() {
    Properties props = new Properties();

    // Core configuration
    props.put("bootstrap.servers", "localhost:9092");
    props.put("group.id", "connect-cluster");
    props.put("client.id", "connect-worker-1");

    // Enhanced security (4.0+)
    props.put("security.protocol", "SASL_SSL");
    props.put("sasl.mechanism", "OAUTHBEARER");
    props.put("sasl.login.callback.handler.class",

"org.apache.kafka.common.security.oauthbearer.secured.OAuthBearerLoginCallbackHand
ler");

    // Improved error handling (4.0+)
    props.put("errors.tolerance", "all");
    props.put("errors.log.enable", "true");
    props.put("errors.deadletterqueue.topic.name", "connect-dlq");
    props.put("errors.deadletterqueue.context.headers.enable", "true");

    // Enhanced monitoring (4.0+)
    props.put("metrics.reporters", "org.apache.kafka.common.metrics.JmxReporter");
    props.put("metrics.jmx.prefix", "kafka.connect");

    // Internal topics with high availability
    props.put("config.storage.topic", "connect-configs");
    props.put("config.storage.replication.factor", 3);

    props.put("offset.storage.topic", "connect-offsets");
    props.put("offset.storage.replication.factor", 3);
    props.put("offset.storage.partitions", 25);

    props.put("status.storage.topic", "connect-status");
    props.put("status.storage.replication.factor", 3);
```

```
        return props;
    }
```

🔗 Additional Resources

📚 Official Documentation

- Kafka Connect Documentation
- Kafka Connect REST API
- Connector Development Guide

🎓 Learning Resources

- Confluent Connect Tutorial
- Building Kafka Connect Connectors
- Kafka Connect Examples

🔌 Connector Ecosystem

- Confluent Hub - Official connector marketplace
- Debezium - CDC connectors
- Kafka Connect File Pulse

📊 Monitoring & Operations

- Kafka Connect Monitoring
- Connect Metrics
- Operational Best Practices

🐛 Troubleshooting

- Common Connect Issues
- Connect FAQ
- Debugging Connect

Last Updated: September 2025
Kafka Version: 4.0.0
Connect Framework: Production-ready since 2.0+

> 💡 **Pro Tip**: Start with well-tested connectors from Confluent Hub, use distributed mode for production, implement comprehensive error handling with DLQ, and monitor connector health continuously. Custom connectors should follow the single responsibility principle and be thoroughly tested.