Интеграция с Kafka для Standard Logger JS

Общее описание

В библиотеку Standard Logger JS добавлена поддержка отправки логов в Apache Kafka. Это позволяет использовать библиотеку в распределенных системах для централизованного сбора и обработки логов.

Что было сделано

1. **Создан транспорт для Kafka**:
   - Добавлен класс `KafkaTransport` в `src/transports/kafka-transport.ts`
   - Реализовано подключение к Kafka, отправка сообщений и управление соединением
2. **Добавлены настройки для Kafka**:
   - Создан интерфейс `KafkaTransportOptions` в `src/interfaces/kafka-options.ts`
   - Определены параметры для подключения, аутентификации и оптимизации
3. **Расширен класс Logger**:
   - Добавлен метод `addKafkaTransport` для настройки отправки логов в Kafka
   - Добавлен метод `close` для корректного закрытия соединений
4. **Создан пример использования**:
   - Пример в `examples/kafka/kafka-example.js` демонстрирует работу с Kafka

Как это работает

Процесс отправки логов

1. При создании лога (`logger.info()`, `logger.error()` и т.д.) формируется сообщение в стандартном формате
2. Если настроен Kafka-транспорт, сообщение отправляется также в `KafkaTransport`
3. `KafkaTransport` проверяет, соответствует ли уровень лога минимальному уровню, указанному в настройках
4. Если уровень подходит, сообщение добавляется в очередь
5. Сообщения отправляются в Kafka:
   - Когда достигнут размер пакета (`batchSize`)
   - Когда срабатывает таймер отправки (`flushIntervalMs`)

- При завершении работы приложения

Ключевые функции

1. **Подключение к Kafka**:

```javascript
async init(): Promise<void> {
    if (this.connected || this.connecting) return;

    this.connecting = true;
    try {
        await this.producer.connect();
        this.connected = true;
        this.startFlushTimer();
    } catch (error) {
        console.error('Failed to connect to Kafka:', error);
    } finally {
        this.connecting = false;
    }
}
```

2. **Отправка сообщений в очередь**:

```javascript
async log(entry: LogEntry): Promise<void> {
    if (!this.shouldLog(entry.level as LogLevel)) return;

    if (!this.connected) {
        await this.init();
    }

    const message: Message = {
        value: JSON.stringify(entry)
    };

    this.messageQueue.push(message);

    if (this.messageQueue.length >= this.batchSize) {
        await this.flush();
    }
}
```

3. **Отправка пакетов сообщений**:

```javascript
private async flush(): Promise<void> {
    if (!this.connected || this.messageQueue.length === 0) return;

    const messages = [...this.messageQueue];
    this.messageQueue = [];

    try {
        await this.producer.send({
            topic: this.topic,
            compression: this.compression,
            messages
        });
    } catch (error) {
        console.error('Failed to send messages to Kafka:', error);
        this.messageQueue = [...messages, ...this.messageQueue];
    }
}
```

4. **Корректное закрытие соединения**:

```javascript
async close(): Promise<void> {
    if (this.flushTimer) {
        clearInterval(this.flushTimer);
        this.flushTimer = null;
    }

    if (this.messageQueue.length > 0) {
        await this.flush();
    }

    if (this.connected) {
        await this.producer.disconnect();
        this.connected = false;
    }
}
```

Как использовать

Установка

bash

```bash
npm install @stepstone/standard-logger
```

Базовое использование

javascript

```javascript
const { createLogger } = require('@stepstone/standard-logger');

// Создаем логгер
const logger = createLogger({
  service: {
    name: 'мой-сервис',
    version: '1.0.0',
    environment: 'production'
  }
});

// Добавляем Kafka транспорт
logger.addKafkaTransport({
  brokers: ['kafka-broker1:9092', 'kafka-broker2:9092'],
  clientId: 'my-service-logger',
  topic: 'application-logs',
  batchSize: 100,
  flushIntervalMs: 3000,
  minLogLevel: 'INFO'  // Отправлять в Kafka только логи с уровнем INFO и выше
});

// Логи теперь будут отправляться в Kafka
logger.info('Приложение запущено');
logger.error('Ошибка подключения к БД', new Error('Connection refused'));

// При завершении работы приложения
process.on('SIGTERM', async () => {
  await logger.close();  // Закрываем соединения
  process.exit(0);
});
```

Параметры настройки

Параметр	Описание	Значение по умолчанию
brokers	Массив адресов Kafka брокеров	Обязательный параметр
clientId	Идентификатор клиента	Обязательный параметр
topic	Тема для отправки логов	Обязательный параметр
compression	Тип сжатия ('gzip', 'snappy', 'lz4', 'zstd')	Без сжатия
batchSize	Размер пакета сообщений	100
flushIntervalMs	Интервал отправки в миллисекундах	5000
minLogLevel	Минимальный уровень логирования ('TRACE', 'DEBUG', 'INFO', 'WARN', 'ERROR', 'CRITICAL')	'TRACE'
ssl	Включить SSL/TLS	false
sasl	Конфигурация SASL аутентификации	undefined

Пример для локального тестирования

1. **Создайте файл** `docker-compose.yml`:

yaml

```yaml
version: '3'
services:
  zookeeper:
    image: confluentinc/cp-zookeeper:latest
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000
    ports:
      - "2181:2181"

  kafka:
    image: confluentinc/cp-kafka:latest
    depends_on:
      - zookeeper
    ports:
      - "9092:9092"
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://localhost:9092
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
      KAFKA_AUTO_CREATE_TOPICS_ENABLE: "true"
```

2. **Запустите контейнеры**:

bash

```bash
docker-compose up -d
```

3. **Запустите пример**:

bash

```bash
npm run example:kafka
```

4. **Проверьте получение сообщений**:

bash

```bash
docker exec -it <ID_контейнера_Kafka> kafka-console-consumer --topic application-logs --bootstr
```

Преимущества

1. **Оптимизация производительности**:
   - Пакетная отправка сообщений
   - Возможность сжатия данных
   - Асинхронная работа, не блокирующая основной поток
2. **Гибкая настройка**:
   - Фильтрация по уровню логирования
   - Настройка размера пакетов и интервала отправки
   - Поддержка SSL и SASL для защищённого подключения
3. **Удобство использования**:
   - Простое подключение через `addKafkaTransport`
   - Сохранение стандартного формата логов
   - Корректное закрытие соединений при завершении работы

Сценарии использования

Микросервисная архитектура

В микросервисной среде интеграция с Kafka позволяет:

- Собирать логи со всех сервисов в одном месте
- Отслеживать запросы через несколько сервисов с помощью идентификаторов трассировки
- Централизованно обрабатывать и анализировать логи

Высоконагруженные системы

Для приложений с большим объемом логов:

- Буферизация сообщений снижает нагрузку на сеть
- Фильтрация по уровню уменьшает объем отправляемых данных
- Сжатие снижает требования к пропускной способности

Мониторинг в продакшн-среде

Для мониторинга работающих систем:

- Оперативное получение информации о критических ошибках
- Возможность создания дашбордов на основе логов

- Аудит действий пользователей и системных событий

Заключение

Интеграция с Kafka значительно расширяет возможности библиотеки Standard Logger JS, делая её подходящей для современных распределенных приложений. Она сохраняет стандартизированный формат логов при этом обеспечивая эффективную и настраиваемую отправку данных в Kafka-кластеры.