ClientKafka

August 18, 2026

ClientKafka

Kind: Class

Source: `packages/microservices/client/client-kafka.ts`

Part of: Microservices

`ClientKafka` is a NestJS microservices client proxy for producing Kafka messages and consuming response messages for request-response patterns. It manages Kafka producer and consumer lifecycle, topic subscriptions, response callbacks, offset commits, and access to the underlying Kafka client.

Extends: `ClientProxy`

Implements: `ClientKafkaProxy`

Methods

Method	Signature	Returns
`subscribeToResponseOf`	`subscribeToResponseOf(pattern: unknown)`	`void`
`close`	`close()`	`Promise<void>`
`connect`	`connect()`	`Promise<Producer>`
`bindTopics`	`bindTopics()`	`Promise<void>`
`createClient`	`createClient()`	`T`
`createResponseCallback`	`createResponseCallback()`	`(payload: EachMessagePayload) => any`
`getConsumerAssignments`	`getConsumerAssignments()`	`void`
`emitBatch`	`emitBatch(pattern: any, data: { messages: TInput[] })`	`Observable<TResult>`

Method	Signature	Returns
`commitOffsets`	`commitOffsets(topicPartitions: TopicPartitionOffsetAndMetadata[])`	`Promise<void>`
`unwrap`	`unwrap()`	`T`
`on`	`on(event: EventKey, callback: EventCallback)`	`void`
`registerConsumerEventListeners`	`registerConsumerEventListeners()`	`void`
`registerProducerEventListeners`	`registerProducerEventListeners()`	`void`
`dispatchBatchEvent`	`dispatchBatchEvent(packets: ReadPacket<{ messages: TInput[] }>)`	`Promise<any>`
`dispatchEvent`	`dispatchEvent(packet: OutgoingEvent)`	`Promise<any>`
`getReplyTopicPartition`	`getReplyTopicPartition(topic: string)`	`string`
`publish`	`publish(partialPacket: ReadPacket, callback: (packet: WritePacket) => any)`	`() => void`
`getResponsePatternName`	`getResponsePatternName(pattern: string)`	`string`
`setConsumerAssignments`	`setConsumerAssignments(data: ConsumerGroupJoinEvent)`	`void`
`initializeSerializer`	`initializeSerializer(options: KafkaOptions['options'])`	`void`
`initializeDeserializer`	`initializeDeserializer(options: KafkaOptions['options'])`	`void`

Properties

Property	Type
`logger`	`any`
`client`	`Kafka
`parser`	`KafkaParser
`initialized`	`Promise

Property	Type
`responsePatterns`	`string[]`
`consumerAssignments`	`{ [key: string]: number }`
`brokers`	`string[]
`clientId`	`string`
`groupId`	`string`
`producerOnlyMode`	`boolean`
`_consumer`	`Consumer
`_producer`	`Producer

Where it refuses work

- `ClientKafka` stops the work with `Error` when `!this._consumer` — “No consumer initialized. Please, call the "connect" method first.”.
- `ClientKafka` stops the work with `Error` when `!this._producer` — “No producer initialized. Please, call the "connect" method first.”.
- `ClientKafka` stops the work with `Error` when `!this.client` — “Not initialized. Please call the "connect" method first.”.
- `ClientKafka` stops the work with `InvalidKafkaClientTopicException` when `isUndefined(minimumPartition)`.
- `ClientKafka` stops the work with an early return when `this.initialized`.
- `ClientKafka` stops the work with an early return when `!this._consumer`.

When something fails

- `ClientKafka` handles failure in 1 place: it turns it into a return value in all 1.

Diagram

```
graph LR
  A[Application Service] --> B[ClientKafka]
  B --> C[Kafka Producer]
  B --> D[Kafka Consumer]
  C --> E[Request / Event Topics]
  E --> F[Kafka Brokers]
  F --> G[Response Topics]
  G --> D
```

```
    D --> H[Response Callback]
    H --> A
```

AI Coding Instructions

- Call `subscribeToResponseOf(pattern)` before `connect()` when using `send()` for Kafka request-response messaging.
- Use `send()` for request-response flows and `emit()` or `emitBatch()` for event-driven, fire-and-forget publishing.
- Always close the client during application shutdown to disconnect the Kafka producer and consumer cleanly.
- Keep Kafka client and consumer settings, especially `clientId`, broker addresses, and consumer `groupId`, consistent with deployment configuration.
- Use `unwrap()` only when direct access to the underlying KafkaJS client is required; prefer the `ClientKafka` APIs for normal messaging.

Relationships

- IMPORTS → `Logger`
- IMPORTS → `loadPackage`
- IMPORTS → `isNil`
- IMPORTS → `isUndefined`

Used by

2 references from 2 files. Each is a place in this repository where the symbol is actually used — go read one rather than trusting an example.

Imported by (2)

- `KafkaController` — `integration/microservices/src/kafka/kafka.controller.ts`:15
- `KafkaConcurrentController` — `integration/microservices/src/kafka-concurrent/kafka-concurrent.controller.ts`:24