ServerKafka

August 18, 2026

ServerKafka

Kind: Class

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

Part of: Microservices

`ServerKafka` is the NestJS microservice transport server responsible for connecting an application to Apache Kafka as both a consumer and producer. It initializes Kafka clients, subscribes to configured message patterns, routes incoming records to registered handlers, and publishes request-response results when required.

Extends: `Server`

Methods

Method	Signature	Returns
`listen`	`listen(callback: (err?: unknown, ...optionalParams: unknown[]) => void)`	`Promise<void>`
`close`	`close()`	`Promise<void>`
`start`	`start(callback: () => void)`	`Promise<void>`
`registerConsumerEventListeners`	`registerConsumerEventListeners()`	`void`
`registerProducerEventListeners`	`registerProducerEventListeners()`	`void`
`createClient`	`createClient()`	`T`
`bindEvents`	`bindEvents(consumer: Consumer)`	`void`
`getMessageHandler`	`getMessageHandler()`	`void`
`getPublisher`	`getPublisher(replyTopic: string, replyPartition: string, correlationId: string, context: KafkaContext)`	`(data: any) => Promise<RecordMetadata[]>`
`handleMessage`	`handleMessage(payload: EachMessagePayload)`	`void`
`unwrap`	`unwrap()`	`T`

Method	Signature	Returns
`on`	`on(event: EventKey, callback: EventCallback)`	`void`
`sendMessage`	`sendMessage(message: OutgoingResponse, replyTopic: string, replyPartition: string	undefined
`assignIsDisposedHeader`	`assignIsDisposedHeader(outgoingResponse: OutgoingResponse, outgoingMessage: Message)`	`void`
`assignErrorHeader`	`assignErrorHeader(outgoingResponse: OutgoingResponse, outgoingMessage: Message)`	`void`
`assignCorrelationIdHeader`	`assignCorrelationIdHeader(correlationId: string, outgoingMessage: Message)`	`void`
`assignReplyPartition`	`assignReplyPartition(replyPartition: string	null
`handleEvent`	`handleEvent(pattern: string, packet: ReadPacket, context: KafkaContext)`	`Promise<any>`
`initializeSerializer`	`initializeSerializer(options: KafkaOptions['options'])`	`void`
`initializeDeserializer`	`initializeDeserializer(options: KafkaOptions['options'])`	`void`

Properties

Property	Type
`transportId`	`TransportId`
`logger`	`any`
`client`	`Kafka
`consumer`	`Consumer
`producer`	`Producer
`parser`	`KafkaParser
`brokers`	`string[]
`clientId`	`string`
`groupId`	`string`

Where it refuses work

- `ServerKafka` stops the work with `Error` when `!this.client` — “Not initialized. Please call the "listen"/"startAllMicroservices" method before accessing…”.
- `ServerKafka` stops the work with an early return when `!handler`, in 2 places.
- `ServerKafka` stops the work with an early return when `!this.consumer`.
- `ServerKafka` stops the work with an early return when `!this.producer`.
- `ServerKafka` stops the work with an early return when `handler?.isEventHandler || !correlationId || !replyTopic`.
- `ServerKafka` stops the work with an early return when `!outgoingResponse.isDisposed`.

When something fails

- `ServerKafka` handles failure in 1 place: it logs it and continues in all 1.

Diagram

```
graph LR
  A[KafkaOptions] --> B[ServerKafka]
  B --> C[createClient]
  C --> D[Kafka Consumer]
  C --> E[Kafka Producer]

  D --> F[registerConsumerEventListeners]
  D --> G[bindEvents]
  G --> H[Incoming Kafka Record]
  H --> I[getMessageHandler]
  I --> J[Registered Message/Event Handler]

  J --> K[getPublisher]
  K --> E
  E --> L[Kafka Response Topic]
```

AI Coding Instructions

- Configure Kafka connection details through `KafkaOptions`, including stable `clientId`, broker addresses, and a unique consumer `groupId`.
- Register message handlers before calling `listen()` so `bindEvents()` can subscribe the consumer to all configured patterns.
- Use event handlers for one-way notifications and request handlers only when callers expect a response published through Kafka.
- Do not manually manage the internal KafkaJS producer or consumer lifecycle; use `listen()` to start connections and `close()` for graceful shutdown.
- Keep handler logic idempotent where possible, since Kafka consumers may receive messages more than once.

Relationships

- IMPORTS → `Logger`
- IMPORTS → `isNil`