EachMessagePayload

August 17, 2026

EachMessagePayload

Kind: Interface

Source: `packages/microservices/external/kafka.interface.ts`

Part of: Microservices

`EachMessagePayload` represents the data delivered to a Kafka consumer handler for each consumed message. It identifies the source topic and partition while providing the raw `KafkaMessage` instance for reading keys, headers, values, and offsets.

Properties

Property	Type
`topic`	`string`
`partition`	`number`
`message`	`KafkaMessage`

Diagram

```
graph LR
  Consumer[Kafka Consumer] --> Payload[EachMessagePayload]
  Payload --> Topic[topic: string]
  Payload --> Partition[partition: number]
  Payload --> Message[message: KafkaMessage]
  Message --> Value[message.value]
  Message --> Headers[message.headers]
  Message --> Offset[message.offset]
```

Usage

```ts
import { EachMessagePayload } from './kafka.interface';

async function handleMessage({
  topic,
  partition,
  message,
}: EachMessagePayload): Promise<void> {
  const value = message.value?.toString('utf8');

  console.log(`Received message from ${topic}[${partition}]`);
  console.log(`Offset: ${message.offset}`);
  console.log(`Payload: ${value}`);
}
```

AI Coding Instructions

- Treat `message.value` as nullable; check for `null` or `undefined` before converting or parsing it.
- Use `topic` and `partition` for logging, tracing, retry diagnostics, and partition-aware processing.
- Read Kafka metadata such as `offset`, `key`, and `headers` from the nested `message` object.
- Avoid assuming message values are JSON; decode the buffer and validate content before parsing.
- Keep message handlers idempotent because Kafka consumers may process a message more than once.

How it works

`EachMessagePayload` is a TypeScript interface representing the argument passed to a Kafka consumer's per-message handler. The surrounding file is a declaration layer intended to represent KafkaJS types rather than Nest-specific logic.
`packages/microservices/external/kafka.interface.ts:1-8`

It requires:

- `topic: string` — the message's topic.
  `packages/microservices/external/kafka.interface.ts:994-996`
- `partition: number` — the topic partition.
  `packages/microservices/external/kafka.interface.ts:995-997`

- `message: KafkaMessage` — the consumed message. `packages/microservices/external/kafka.interface.ts:996-997` `KafkaMessage` is either a message-set entry or a record-batch entry; both include nullable `key` and `value`, a string `timestamp`, numeric `attributes`, and a string `offset`. Record-batch entries have `headers`, while message-set entries declare that `headers` cannot be present. `packages/microservices/external/kafka.interface.ts:718-738`
- `heartbeat(): Promise<void>` — an asynchronous method whose declared result has no value. `packages/microservices/external/kafka.interface.ts:997-999`
- `pause(): () => void` — a method returning a zero-argument function with no declared return value. `packages/microservices/external/kafka.interface.ts:998-1000`

`EachMessageHandler` accepts this payload and must return `Promise<void>`. `ConsumerRunConfig.eachMessage` is optional and is typed as that handler, and `Consumer.run()` accepts this run configuration. `packages/microservices/external/kafka.interface.ts:1025-1036` `packages/microservices/external/kafka.interface.ts:1043-1049`

In Nest's Kafka client, `bindTopics()` passes `createResponseCallback()` as `eachMessage` when starting the consumer. `packages/microservices/client/client-kafka.ts:196-214` That callback mutates `payload.message` by assigning the payload's `topic` and `partition` before parsing it; it then ignores messages without a correlation-ID header or without a matching routing callback. `packages/microservices/client/client-kafka.ts:227-255`

In Nest's Kafka server, `bindEvents()` similarly installs `getMessageHandler()` as `eachMessage`, and that handler forwards the payload to `handleMessage()`. `packages/microservices/server/server-kafka.ts:165-184` `handleMessage()` also assigns `topic` and `partition` onto `payload.message` before parsing it, then reads request-related headers and deserializes the parsed message. `packages/microservices/server/server-kafka.ts:202-215`

The interface itself contains no implementation, runtime validation, thrown errors, or side effects; it only declares the required payload shape and method signatures. `packages/microservices/external/kafka.interface.ts:994-1000`