PartitionerArgs

August 18, 2026

PartitionerArgs

Kind: Interface

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

Part of: Microservices

`PartitionerArgs` describes the inputs provided to a Kafka partitioning strategy when selecting a partition for an outgoing message. It combines the target topic, available partition metadata, and the message being published so custom partitioners can make deterministic routing decisions.

Properties

Property	Type
`topic`	`string`
`partitionMetadata`	`PartitionMetadata[]`
`message`	`Message`

Diagram

```
graph LR
  P[PartitionerArgs] --> T[topic: string]
  P --> PM[partitionMetadata: PartitionMetadata[]]
  P --> M[message: Message]

  T --> K[Kafka topic]
  PM --> AP[Available partitions]
  M --> PS[Custom partitioning strategy]
  AP --> PS
  K --> PS
```

Usage

```ts
import type { PartitionerArgs } from '@nestjs/microservices';

function selectPartition({
  topic,
  partitionMetadata,
  message,
}: PartitionerArgs): number {
  const key = message.key?.toString() ?? topic;

  let hash = 0;
  for (const character of key) {
    hash = (hash * 31 + character.charCodeAt(0)) >>> 0;
  }

  const availablePartitions = partitionMetadata.filter(
    (partition) => partition.leader !== -1,
  );

  return hash % availablePartitions.length;
}
```

AI Coding Instructions

- Use `message.key` when possible to ensure messages with the same key are consistently routed to the same partition.
- Filter or account for unavailable partitions in `partitionMetadata` before returning a partition index.
- Return a valid partition index based on the available metadata; avoid hardcoding partition counts.
- Keep custom partitioning deterministic, since non-deterministic routing can break ordering guarantees.
- Ensure the partitioner is registered through the Kafka client or producer configuration where custom partitioning is supported.

How it works

`PartitionerArgs` is an exported TypeScript interface representing the argument passed to the function returned by an `ICustomPartitioner`. The surrounding file is explicitly a KafkaJS type representation and says it must not contain NestJS logic.

`packages/microservices/external/kafka.interface.ts:1-8`

`packages/microservices/external/kafka.interface.ts:129-135`

It requires three fields:

- `topic` : a `string` . `packages/microservices/external/kafka.interface.ts:129-131`
- `partitionMetadata` : an array of `PartitionMetadata` , whose entries include a partition ID, leader, replicas, in-sync replicas, and an optional offline-replica list.
  `packages/microservices/external/kafka.interface.ts:131`
  `packages/microservices/external/kafka.interface.ts:151-158`
- `message` : a `Message` with a required `value` ; it may also contain a key, explicit partition, headers, and timestamp.
  `packages/microservices/external/kafka.interface.ts:132`
  `packages/microservices/external/kafka.interface.ts:121-127`

A custom partitioner has the shape `() => (args: PartitionerArgs) => number` ; therefore, its returned function receives this object and returns a numeric partition selection. `ProducerConfig.createPartitioner` optionally accepts that custom-partitioner factory.
`packages/microservices/external/kafka.interface.ts:110-119`
`packages/microservices/external/kafka.interface.ts:135-137`

For example, an integration controller destructures `message` from `PartitionerArgs` and returns the numeric value of the `toPartition` message header; it assigns that function to `producer.createPartitioner` . `integration/microservices/src/kafka-concurrent/kafka-concurrent.controller.ts:18-22` `integration/microservices/src/kafka-concurrent/kafka-concurrent.controller.ts:39-41`

`PartitionerArgs` itself declares no methods, validation, thrown errors, or runtime side effects. `packages/microservices/external/kafka.interface.ts:129-133`