NatsOptions

August 17, 2026

NatsOptions

Kind: Interface

Source: `packages/microservices/interfaces/microservice-configuration.interface.ts`

Part of: Microservices

`NatsOptions` defines the configuration required to run a NestJS microservice using the `Transport.NATS` transport. It combines Nest-specific settings such as serializers, deserializers, and queue groups with NATS client connection, authentication, reconnection, TLS, and request behavior options.

Properties

Property	Type
`transport`	`Transport.NATS`
`options`	`{ headers?: Record<string, string>; authenticator?: any; debug?: boolean; ignoreClusterUpdates?: boolean; inboxPrefix?: string; encoding?: string; name?: string; user?: string; pass?: string; maxPingOut?: number; maxReconnectAttempts?: number; reconnectTimeWait?: number; reconnectJitter?: number; reconnectJitterTLS?: number; reconnectDelayHandler?: any; servers?: string[]`

Diagram

```
graph LR
  App[Nest Application] --> Config[NatsOptions]
  Config --> Transport[Transport.NATS]
  Config --> Connection[NATS Connection Settings]
  Config --> Auth[Authentication / TLS]
  Config --> Reconnect[Reconnect & Ping Settings]
  Config --> Messaging[Queue, Headers, Request Settings]
  Config --> Serialization[Serializer / Deserializer]

  Connection --> NATS[NATS Server or Cluster]
  Auth --> NATS
```

```
  Reconnect --> NATS
  Messaging --> NATS
  Serialization --> NATS
```

Usage

```ts
import { NestFactory } from '@nestjs/core';
import { Transport, type NatsOptions } from '@nestjs/microservices';
import { AppModule } from './app.module';

async function bootstrap() {
  const natsOptions: NatsOptions = {
    transport: Transport.NATS,
    options: {
      servers: ['nats://localhost:4222'],
      queue: 'orders-service',
      name: 'orders-microservice',
      user: process.env.NATS_USER,
      pass: process.env.NATS_PASSWORD,
      reconnect: true,
      maxReconnectAttempts: 10,
      reconnectTimeWait: 1_000,
      pingInterval: 30_000,
      headers: {
        'x-service-name': 'orders-service',
      },
    },
  };

  const app = await NestFactory.createMicroservice(AppModule, natsOptions);
  await app.listen();
}

bootstrap();
```

AI Coding Instructions

- Always set `transport: Transport.NATS`; place NATS client and Nest transport configuration inside `options`.
- Configure `servers` with one or more NATS endpoints, especially when connecting to a clustered deployment.
- Use environment variables or secret management for credentials, tokens, JWTs, NKeys, and TLS configuration; do not hardcode sensitive values.
- Set a stable `queue` when multiple service instances should share subscriptions through a NATS queue group.

- Configure `serializer` and `deserializer` consistently across communicating services when using custom message formats.

How it works

`NatsOptions` is a public TypeScript configuration interface for selecting the NATS microservice transport. It is one member of the `MicroserviceOptions` union, and its optional `transport` property is restricted to `Transport.NATS`. packages/microservices/interfaces/microservice-configuration.interface.ts:25-33 packages/microservices/interfaces/microservice-configuration.interface.ts:170-175 `Transport.NATS` is the NATS enum member. packages/microservices/enums/transport.enum.ts:1-9

All declared configuration is optional: both `transport` and the nested `options` object may be omitted. packages/microservices/interfaces/microservice-configuration.interface.ts:173-216 When a client is created through `ClientProxyFactory` with `Transport.NATS`, the factory substitutes `{}` when its top-level `options` is absent and constructs `ClientNats`. packages/microservices/client/client-proxy-factory.ts:48-57

Connection options

The `options` object declares NATS connection-related fields including `servers`, authentication-related fields (`authenticator`, `user`, `pass`, `token`, `nkey`, `userJWT`, `nonceSigner`, `userCreds`, and `tokenHandler`), TLS, reconnect controls, ping controls, `timeout`, and other NATS flags. Several callback-like or credential fields are typed as `any`; `servers` accepts either one string or a string array. packages/microservices/interfaces/microservice-configuration.interface.ts:175-215

It also has an index signature, `[key: string]: any`, so TypeScript permits additional string-keyed option values. packages/microservices/interfaces/microservice-configuration.interface.ts:213-215

Both `ClientNats` and `ServerNats` load the optional `nats` package and call its `connect` function with a default `servers` value of `nats://localhost:4222`, followed by a spread of this options object. Consequently, an `options.servers` value overrides that default in these paths. packages/microservices/client/client-nats.ts:38-43 packages/microservices/client/client-nats.ts:69-75 packages/microservices/server/server-nats.ts:50-59 packages/microservices/server/server-nats.ts:126-132 packages/microservices/constants.ts:7

Nest-specific options and behavior

- `queue` is used by `ServerNats` as the default queue when it subscribes each registered message-handler channel. A handler-level `extras.queue` takes precedence when present. packages/microservices/interfaces/microservice-configuration.interface.ts:195-198 packages/microservices/server/server-nats.ts:82-96
- `headers` is merged into client-published request and event headers. Existing serialized-message header keys are retained; configured headers are added only when that key is absent. packages/microservices/interfaces/microservice-configuration.interface.ts:175-177 packages/microservices/client/client-nats.ts:223-227 packages/microservices/client/client-nats.ts:236-245 packages/microservices/client/client-nats.ts:262-275
- `inboxPrefix` is passed to `nats.createInbox()` for client request/reply publishing. packages/microservices/interfaces/microservice-configuration.interface.ts:179-182 packages/microservices/client/client-nats.ts:209-227
- `serializer` and `deserializer` accept objects implementing `serialize()` and `deserialize()` respectively. packages/microservices/interfaces/microservice-configuration.interface.ts:196-198 packages/microservices/interfaces/serializer.interface.ts:10-12 packages/microservices/interfaces/deserializer.interface.ts:10-15 The client defaults to `NatsRecordSerializer` and `NatsResponseJSONDeserializer`; the server defaults to `NatsRecordSerializer` and `NatsRequestJSONDeserializer` when these fields are absent. packages/microservices/client/client-nats.ts:253-260 packages/microservices/server/server-nats.ts:281-288
- When `debug` is truthy, both client and server log NATS `pingTimer` status updates at debug level. packages/microservices/interfaces/microservice-configuration.interface.ts:176-179 packages/microservices/client/client-nats.ts:126-132 packages/microservices/server/server-nats.ts:225-231
- On server shutdown, `gracefulShutdown` causes all tracked subscriptions to unsubscribe, then waits for `gracePeriod`; if `gracePeriod` is absent, the wait is `10000` milliseconds, before closing the NATS client. Without `gracefulShutdown`, the server closes the client without that unsubscribe-and-wait branch. packages/microservices/interfaces/microservice-configuration.interface.ts:213-214 packages/microservices/server/server-nats.ts:99-124 packages/microservices/constants.ts:65

The observed client and server connection paths spread the options directly into `nats.connect`; they contain no field-by-field validation before that call.
packages/microservices/client/client-nats.ts:69-75
packages/microservices/server/server-nats.ts:126-132