ServerNats

August 18, 2026

ServerNats

Kind: Class

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

Part of: Microservices

`ServerNats` is the NATS transport server implementation for NestJS microservices. It connects to a NATS broker, subscribes registered message handlers to their patterns, routes incoming requests or events to those handlers, and manages connection lifecycle and status updates.

Extends: `Server`

Methods

Method	Signature	Returns
`listen`	`listen(callback: (err?: unknown, ...optionalParams: unknown[]) => void)`	`void`
`start`	`start(callback: (err?: unknown, ...optionalParams: unknown[]) => void)`	`void`
`bindEvents`	`bindEvents(client: Client)`	`void`
`close`	`close()`	`void`
`createNatsClient`	`createNatsClient()`	`Promise<Client>`
`getMessageHandler`	`getMessageHandler(channel: string)`	`Function`
`handleMessage`	`handleMessage(channel: string, natsMsg: NatsMsg)`	`void`
`getPublisher`	`getPublisher(natsMsg: NatsMsg, id: string, ctx: NatsContext)`	`void`
`handleStatusUpdates`	`handleStatusUpdates(client: Client)`	`void`

Method	Signature	Returns
`unwrap`	`unwrap()`	`T`
`on`	`on(event: EventKey, callback: EventCallback)`	`void`
`initializeSerializer`	`initializeSerializer(options: NatsOptions['options'])`	`void`
`initializeDeserializer`	`initializeDeserializer(options: NatsOptions['options'])`	`void`

Properties

Property	Type
`transportId`	`TransportId`
`statusEventEmitter`	`any`

Where it refuses work

- `ServerNats` stops the work with `Error` when `!this.natsClient` — “Not initialized. Please call the "listen"/"startAllMicroservices" method before accessing…”.
- `ServerNats` stops the work with an early return when `!this.natsClient`.
- `ServerNats` stops the work with an early return when `error`.
- `ServerNats` stops the work with an early return when `isUndefined((message as IncomingRequest).id)`.
- `ServerNats` stops the work with an early return when `natsMsg.reply`.

When something fails

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

Diagram

```
graph LR
  A[Nest Microservice] --> B[ServerNats]
  B --> C[createNatsClient]
  C --> D[NATS Broker]
  B --> E[bindEvents]
  E --> F[Registered Message Handlers]
  D --> G[Incoming NATS Message]
```

```
  G --> H[getMessageHandler / handleMessage]
  H --> F
  B --> I[getPublisher]
  I --> D
  B --> J[close]
  J --> D
```

Usage

```
import { NestFactory } from '@nestjs/core';
import { ServerNats } from '@nestjs/microservices';
import { AppModule } from './app.module';

async function bootstrap() {
  const natsServer = new ServerNats({
    servers: ['nats://localhost:4222'],
    queue: 'orders-service',
  });

  const app = await NestFactory.createMicroservice(AppModule, {
    strategy: natsServer,
  });

  await app.listen();

  // Access the underlying NATS client when transport-specific APIs are needed.
  const client = natsServer.unwrap();
  console.log('Connected to NATS:', Boolean(client));
}

bootstrap();
```

AI Coding Instructions

- Use `ServerNats` as a custom NestJS microservice transport strategy; let Nest register handlers before calling `listen()`.
- Register message patterns through Nest decorators such as `@MessagePattern()` and `@EventPattern()` rather than manually subscribing through the underlying client.
- Keep NATS connection settings, queue groups, and serializer/deserializer options in the transport configuration passed to the constructor.
- Call `close()` during application shutdown when managing the server lifecycle manually.
- Use `unwrap()` only for NATS-client-specific functionality; avoid bypassing Nest message handling for normal request and event processing.

Relationships

- IMPORTS → `isObject`
- IMPORTS → `isUndefined`

Used by

1 reference from 1 file. Each is a place in this repository where the symbol is actually used — go read one rather than trusting an example.

Imported by (1)

- `NatsSubscriber` — `sample/03-microservices/src/common/strategies/nats.strategy.ts`:3