ClientProxyFactory

August 18, 2026

ClientProxyFactory

Kind: Class

Source: `packages/microservices/client/client-proxy-factory.ts`

Part of: Microservices

`ClientProxyFactory` creates microservice client proxies based on the configured transport strategy, such as TCP, Redis, NATS, gRPC, or Kafka. It centralizes transport-specific client selection so application code can interact with a common `ClientProxy` interface for sending messages and emitting events.

Methods

Method	Signature	Returns
`create`	`create(clientOptions: { transport: Transport.GRPC } & ClientOptions)`	`ClientGrpcProxy`
`create`	`create(clientOptions: { transport: Transport.KAFKA } & ClientOptions)`	`ClientKafkaProxy`
`create`	`create(clientOptions: ClientOptions)`	`ClientProxy`
`create`	`create(clientOptions: CustomClientOptions)`	`ClientProxy`
`create`	`create(clientOptions: ClientOptions`	CustomClientOptions)`

Diagram

```
graph LR
  A[Client Options] --> B[ClientProxyFactory.create]
  B --> C{Transport Type}
  C -->|TCP / Redis / NATS / RMQ / MQTT| D[ClientProxy]
  C -->|gRPC| E[ClientGrpcProxy]
  C -->|Kafka| F[ClientKafkaProxy]
  D --> G[Microservice Broker or Server]
```

```
  E --> G
  F --> G
```

Usage

```ts
import { ClientProxyFactory, Transport } from '@nestjs/microservices';

const client = ClientProxyFactory.create({
  transport: Transport.TCP,
  options: {
    host: 'localhost',
    port: 3001,
  },
});

// Request-response communication
const result = await client.send('users.findOne', { id: '123' }).toPromise();

// Event-based communication
client.emit('users.created', {
  id: '123',
  email: 'user@example.com',
});
```

AI Coding Instructions

- Use `ClientProxyFactory.create()` when creating clients dynamically outside of Nest dependency injection.
- Match the client `transport` and connection options with the corresponding microservice server configuration.
- Use `send()` for request-response patterns and `emit()` for fire-and-forget event patterns.
- Call `connect()` explicitly when eager connection validation is required; otherwise, clients connect lazily on first use.
- Prefer injected clients registered through `ClientsModule` for application services, reserving this factory for runtime or infrastructure-level client creation.

Used by

7 references from 7 files. Each is a place in this repository where the symbol is actually used — go read one rather than trusting an example.

Imported by (7)

- `DisconnectedClientController` — `integration/microservices/src/disconnected.controller.ts` :12
- `NatsController` — `integration/microservices/src/nats/nats.controller.ts` :19
- `RMQFanoutExchangeProducerController` — `integration/microservices/src/rmq/fanout-exchange-producer-rmq.controller.ts` :9
- `RMQBroadcastController` — `integration/microservices/src/rmq/rmq-broadcast.controller.ts` :11
- `RMQController` — `integration/microservices/src/rmq/rmq.controller.ts` :16
- `RMQTopicExchangeController` — `integration/microservices/src/rmq/topic-exchange-rmq.controller.ts` :12
- `HttpController` — `integration/scopes/src/msvc/http.controller.ts` :4