Client

August 18, 2026

Client

Kind: Function

Source: `packages/microservices/decorators/client.decorator.ts`

Part of: Microservices

Attaches the `ClientProxy` instance to the given property

`Client()` is a property decorator that attaches a `ClientProxy` instance to a class property. It registers the supplied microservice client options so the application can create and use a transport-specific proxy for sending messages or emitting events to another service.

Signature

```
function Client(metadata: ClientOptions): PropertyDecorator
```

Parameters

Name	Type
`metadata`	`ClientOptions`

Returns: `PropertyDecorator`

Diagram

```
graph LR
  A[Service class property] --> B["@Client(clientOptions)"]
  B --> C[ClientProxy configuration metadata]
  C --> D[ClientProxy instance]
  D --> E[Remote microservice transport]
```

Usage

```
import { Injectable } from '@nestjs/common';
import { Client, ClientProxy, Transport } from '@nestjs/microservices';

@Injectable()
export class OrdersService {
  @Client({
    transport: Transport.TCP,
    options: {
      host: 'localhost',
      port: 3001,
    },
  })
  private readonly inventoryClient: ClientProxy;

  async reserveInventory(productId: string, quantity: number) {
    return this.inventoryClient.send(
      { cmd: 'reserve_inventory' },
      { productId, quantity },
    );
  }
}
```

AI Coding Instructions

- Apply `@Client()` only to class properties intended to hold a `ClientProxy`; declare the property type as `ClientProxy`.
- Provide transport-specific configuration through the decorator options, including required connection details such as host, port, queue, or broker URLs.
- Use `client.send()` for request-response messaging and `client.emit()` for event-based communication.
- Ensure the target microservice is configured with the same transport and compatible message patterns.
- Avoid manually constructing client proxies when decorator-based property injection is sufficient; keep client configuration close to the consuming service.

Used by

11 references from 11 files. Each is a place in this repository where the symbol is actually used — go read one rather than trusting an example.

Imported by (11)

- `AppController` — `integration/microservices/src/app.controller.ts`:20
- `GrpcController` — `integration/microservices/src/grpc/grpc.controller.ts`:21
- `AdvancedGrpcController` — `integration/microservices/src/grpc-advanced/advanced.grpc.controller.ts`:14
- `KafkaController` — `integration/microservices/src/kafka/kafka.controller.ts`:15
- `KafkaConcurrentController` — `integration/microservices/src/kafka-concurrent/kafka-concurrent.controller.ts`:24
- `MqttBroadcastController` — `integration/microservices/src/mqtt/mqtt-broadcast.controller.ts`:11
- `MqttController` — `integration/microservices/src/mqtt/mqtt.controller.ts`:16
- `NatsBroadcastController` — `integration/microservices/src/nats/nats-broadcast.controller.ts`:11

…and 3 more.