# NestFactoryStatic

August 18, 2026

## NestFactoryStatic

**Kind:** Class

**Source:** `packages/core/nest-factory.ts`

**Part of:** Core

`NestFactoryStatic` is the core factory responsible for bootstrapping Nest applications from a root module. It creates HTTP applications, standalone application contexts, and microservices while coordinating dependency scanning, container initialization, and adapter setup. In typical applications, it is accessed through the exported `NestFactory` instance.

### Methods

| Method | Signature | Returns |
| --- | --- | --- |
| `create` | `create(module: IEntryNestModule, options: NestApplicationOptions)` | `Promise<T>` |
| `create` | `create(module: IEntryNestModule, httpAdapter: AbstractHttpAdapter, options: NestApplicationOptions)` | `Promise<T>` |
| `create` | `create(moduleCls: IEntryNestModule, serverOrOptions: AbstractHttpAdapter | NestApplicationOptions, options: NestApplicationOptions)` |
| `createMicroservice` | `createMicroservice(moduleCls: IEntryNestModule, options: NestMicroserviceOptions & T)` | `Promise<INestMicroservice>` |
| `createApplicationContext` | `createApplicationContext(moduleCls: IEntryNestModule, options: NestApplicationContextOptions)` | `Promise<INestApplicationContext>` |

### Where it refuses work

- `NestFactoryStatic` stops the work with an early return when `isFunction(receiver[prop])`, in 2 places.

- `NestFactoryStatic` stops the work with an early return when `!(prop in receiver)`.
- `NestFactoryStatic` stops the work with an early return when `!options`.
- `NestFactoryStatic` stops the work with an early return when `!(prop in receiver) && prop in adapter`.

## When something fails

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

## Diagram

```
graph LR
  RootModule[Root Module] --> Factory[NestFactoryStatic]
  Factory --> HTTP[create()]
  Factory --> Microservice[createMicroservice()]
  Factory --> Context[createApplicationContext()]
  HTTP --> App[INestApplication]
  Microservice --> Micro[INestMicroservice]
  Context --> Standalone[INestApplicationContext]
```

## Usage

```
import { Module } from '@nestjs/common';
import { NestFactory } from '@nestjs/core';

@Module({})
class AppModule {}

async function bootstrap() {
  // Create an HTTP application.
  const app = await NestFactory.create(AppModule);
  await app.listen(3000);

  // Create a standalone dependency-injection context.
  const context = await NestFactory.createApplicationContext(AppModule);

  // Create a microservice when a transport configuration is available.
  // const microservice = await NestFactory.createMicroservice(AppModule, {
  //   transport: Transport.TCP,
  // });
  // await microservice.listen();
}

bootstrap();
```

## AI Coding Instructions

- Use `create()` for HTTP-based Nest applications, `createMicroservice()` for transport-driven services, and `createApplicationContext()` for CLI jobs, workers, or scripts without an HTTP server.
- Pass the root module as the first argument; ensure its imports, providers, and controllers are configured before application bootstrap.
- Await factory calls before accessing application APIs such as `listen()`, `get()`, `connectMicroservice()`, or `close()`.
- Prefer the public `NestFactory` export rather than instantiating or depending directly on `NestFactoryStatic`.
- Ensure standalone contexts and microservices are explicitly closed during tests, scripts, and graceful shutdown flows.