Pregunta:

Hola Gemini, Nestjs me toma las variables de entorno del fichero .env y he creado otro fichero llamado .env.production para las variables de entorno de producción, sin embargo cuando ejecuto un npm run start:prod me sigue tomando la variables del fichero .env, en vez de el fichero de .env.production, ¿qué estoy haciendo mal?

Respuesta:

## El misterio del .env.production: Por qué NestJS ignora tu fichero de producción y cómo solucionarlo

¡Hola! Es un problema bastante común al configurar diferentes entornos en una aplicación de NestJS. La razón por la que tu aplicación sigue tomando las variables del fichero .env en lugar del .env.production cuando ejecutas npm run start:prod se debe a que NestJS, por defecto, no sabe que debe buscar un fichero de entorno diferente para el entorno de producción. Es necesario configurarlo explícitamente.

A continuación, te explico detalladamente qué está sucediendo y cómo puedes solucionarlo paso a paso.

### La raíz del problema: El ConfigModule de NestJS

NestJS utiliza el módulo @nestjs/config para gestionar las variables de entorno. Cuando lo inicializas en tu app.module.ts, si no le indicas lo contrario, buscará un único fichero llamado .env en la raíz de tu proyecto.

Para que tu aplicación sea capaz de seleccionar entre .env y .env.production (o cualquier otro fichero de entorno), necesitas indicárselo de forma dinámica. La práctica recomendada es utilizar la variable de entorno NODE\_ENV.

### La solución paso a paso

Sigue estos pasos para configurar correctamente tu proyecto de NestJS y que utilice el fichero .env.production en el entorno de producción:

#### 1. Instala cross-env (Recomendado)

Para asegurar que tus scripts de npm funcionen de manera consistente en diferentes sistemas operativos (como Windows, macOS y Linux), es una buena práctica utilizar el paquete cross-env. Este paquete te permite establecer variables de entorno de forma universal.

Si no lo tienes instalado, puedes añadirlo a tus dependencias de desarrollo con el siguiente comando:

Bash

npm install --save-dev cross-env

#### 2. Modifica tu package.json

El siguiente paso es asegurarte de que tu script start:prod establece la variable de entorno NODE\_ENV a production antes de ejecutar la aplicación.

Abre tu fichero package.json y modifica la sección de "scripts" de la siguiente manera:

JSON

"scripts": {  
 "start": "nest start",  
 "start:dev": "cross-env NODE\_ENV=development nest start --watch",  
 "start:prod": "cross-env NODE\_ENV=production node dist/main"  
}

**Nota:** Es importante que el script start:prod ejecute el código compilado de JavaScript en la carpeta dist directamente con node. El comando nest start está más orientado al desarrollo.

#### 3. Configura el ConfigModule en tu app.module.ts

Ahora, necesitas indicarle a NestJS que cargue el fichero .env correspondiente en función del valor de NODE\_ENV.

Abre tu fichero app.module.ts y modifica la importación del ConfigModule de la siguiente manera:

TypeScript

import { Module } from '@nestjs/common';  
import { ConfigModule } from '@nestjs/config';  
import { AppController } from './app.controller';  
import { AppService } from './app.service';  
  
@Module({  
 imports: [  
 ConfigModule.forRoot({  
 envFilePath: `.env.${process.env.NODE\_ENV}`,  
 isGlobal: true,  
 }),  
 ],  
 controllers: [AppController],  
 providers: [AppService],  
})  
export class AppModule {}

Con esta configuración, le estás diciendo a NestJS: "Busca un fichero .env seguido del valor que tenga la variable de entorno NODE\_ENV".

* Si NODE\_ENV es production, buscará .env.production.
* Si NODE\_ENV es development (como en el script start:dev), buscará .env.development.
* Si NODE\_ENV no está definida, intentará buscar un fichero llamado .env.undefined, lo cual probablemente no exista. Por eso es importante tener un fichero .env como fallback o asegurar que NODE\_ENV siempre esté definida.

#### 4. Crea tus ficheros de entorno

Asegúrate de tener los siguientes ficheros en la raíz de tu proyecto:

* **.env**: Este fichero puede contener las variables por defecto o las de desarrollo si no usas un fichero específico para ello.
* **.env.production**: Este fichero contendrá las variables de entorno específicas para el entorno de producción (por ejemplo, la URL de la base de datos de producción, las claves de API de producción, etc.).

**Ejemplo de .env (para desarrollo):**

DATABASE\_URL=postgres://user:password@localhost:5432/development\_db  
API\_KEY=my\_development\_api\_key

**Ejemplo de .env.production:**

DATABASE\_URL=postgres://user:password@production-db-url.com:5432/production\_db  
API\_KEY=my\_production\_api\_key

### Resumen del flujo

1. Cuando ejecutas npm run start:prod.
2. cross-env establece la variable de entorno NODE\_ENV a production.
3. Se ejecuta node dist/main, que inicia tu aplicación de NestJS ya compilada.
4. El ConfigModule, gracias a la configuración en app.module.ts, lee el valor de process.env.NODE\_ENV (que ahora es "production").
5. Construye la ruta del fichero de entorno como .env.production y carga las variables desde ese fichero.

Con estos cambios, tu aplicación de NestJS cargará las variables de entorno correctas para cada entorno, permitiéndote tener una configuración limpia y segura para desarrollo y producción.

**Fuentes**

1. <https://github.com/lowcode-scaffold/lowcode-materials>