Pregunta:

Cuando ejecuto npm run dev en next.js me busca el puerto libre que haya para el servidor y me lo pone en marcha, pero cuando ejecuto npm run start no me lo permite por buscar exclusivamente el puerto 3000 y estar ocupado. ¿Cómo configuro next.js para que comience con un puerto suministrado desde las variables de entorno?

Respuesta:

¡Hola! Este es un comportamiento muy habitual en Next.js y tiene una explicación lógica. El comando npm run dev está diseñado para la flexibilidad durante el desarrollo, mientras que npm run start está pensado para un entorno de producción, que suele ser más estático y predecible.

A continuación te explico por qué ocurre esto y cómo puedes configurar Next.js para que utilice el puerto que tú le indiques a través de las variables de entorno, tanto en desarrollo como en producción.

### ¿Por qué npm run dev y npm run start se comportan de forma diferente?

* **npm run dev (next dev):** Inicia el servidor de desarrollo. Si el puerto por defecto (3000) está ocupado, Next.js es lo suficientemente inteligente como para buscar automáticamente el siguiente puerto libre (3001, 3002, etc.). Esto es muy útil para evitar conflictos cuando tienes varios proyectos corriendo a la vez en tu máquina.
* **npm run start (next start):** Inicia el servidor de producción. Este comando se ejecuta después de haber construido tu aplicación con npm run build. Está optimizado para el rendimiento y la estabilidad, por lo que asume una configuración fija. No busca puertos alternativos y, si el puerto 3000 está ocupado, simplemente falla.

### La solución: Configurar el puerto con variables de entorno

La forma correcta de especificar un puerto para los comandos de Next.js es a través de la variable de entorno PORT o pasándolo como un argumento en el script.

Aquí tienes la solución paso a paso:

#### 1. Modifica tus scripts en package.json

La manera más clara y recomendada es modificar los scripts en tu fichero package.json para que lean una variable de entorno. Para asegurar la compatibilidad entre diferentes sistemas operativos (Windows, macOS, Linux), es muy recomendable usar el paquete cross-env, tal como en la solución de NestJS.

Si no lo tienes, instálalo:

Bash

npm install --save-dev cross-env

Ahora, ajusta tu package.json:

JSON

"scripts": {  
 "dev": "next dev",  
 "build": "next build",  
 "start": "next start -p $PORT",  
 "lint": "next lint"  
}

Actualización:

Una forma más moderna y compatible con cross-env es definir la variable de entorno directamente en el script start. El flag -p es el método preferido por Next.js para el comando start.

Modifica tu script start en package.json de la siguiente manera:

JSON

"scripts": {  
 "dev": "next dev",  
 "build": "next build",  
 "start": "next start -p ${PORT:-3000}", // Opción recomendada  
 "lint": "next lint"  
}

**¿Qué hace -p ${PORT:-3000}?**

* Le dice al comando next start que use el puerto (-p) especificado en la variable de entorno PORT.
* Si la variable PORT **no está definida**, utiliza el valor 3000 como fallback.

#### 2. Crea tus ficheros de variables de entorno

Next.js tiene un sistema integrado excelente para manejar variables de entorno. No necesitas instalar paquetes adicionales como dotenv. Simplemente crea los ficheros adecuados en la raíz de tu proyecto.

* **.env.local**: Para tus variables locales. **No se sube a Git**. Ideal para la mayoría de los casos de desarrollo.
* **.env.production**: Para las variables del entorno de producción. Se carga cuando ejecutas npm run build y npm run start.
* **.env.development**: Para las variables del entorno de desarrollo. Se carga cuando ejecutas npm run dev.

**Ejemplo:**

1. **Crea el fichero .env.local** (para desarrollo si quieres un puerto específico):  
   PORT=3001
2. **Crea el fichero .env.production** (para producción):  
   PORT=8080

#### 3. ¿Cómo funciona todo junto?

Con la configuración anterior:

* **Cuando ejecutes npm run dev:**
  + Next.js cargará las variables de .env.development o .env.local.
  + El comando next dev también respeta la variable PORT. Si defines PORT=3001 en .env.local, intentará arrancar en el puerto 3001. Si está ocupado, buscará el siguiente libre.
* **Cuando ejecutes npm run start:**
  + Primero, asegúrate de haber construido la aplicación con npm run build.
  + Next.js cargará las variables de .env.production.
  + El script start ejecutará next start -p ${PORT:-3000}.
  + Si definiste PORT=8080 en .env.production, el comando se convertirá en next start -p 8080 y el servidor de producción se iniciará en ese puerto.
  + Si no tienes un .env.production o la variable PORT no está definida, arrancará en el puerto 3000.

### Resumen de la mejor configuración

1. **package.json**:  
   JSON  
   "scripts": {  
    "dev": "next dev",  
    "build": "next build",  
    "start": "next start -p ${PORT:-3000}",  
    "lint": "next lint"  
   }
2. **.env.production** (Ejemplo para un servidor que necesita correr en el puerto 5000):  
   PORT=5000
3. **.env.local** (Ejemplo para desarrollo en el puerto 4000):  
   PORT=4000

Ahora, tu comando npm run start será flexible y tomará el puerto que le hayas configurado en tus variables de entorno, solucionando el problema del puerto 3000 ocupado de una manera robusta y predecible.