Pregunta:

He seguido los pasos que me das incluyendo poner el script start a "next start -p ${PORT:-3000}, creando el fichero .env.production y el .env.local y no carga la información de los ficheros de variables de entorno. También he cambiado el script de start a "cross-env NODE\_ENV=production next start -p ${PORT:-3000} y sigue sin cargar la información de esos ficheros.

¿Hay algún cambio en el next.config.ts u otro similar de next que refleje qué fichero de variables de entorno cargar? ¿o cómo solucionarlo?

Respuesta:

Entiendo tu frustración. Es un problema común cuando las variables de entorno no se cargan como se espera, especialmente en entornos de producción. Next.js tiene un sistema robusto para esto, pero a veces hay detalles que se nos escapan.

Aquí te explico las claves y cómo solucionarlo, centrándome en las mejores prácticas de Next.js:

## Cómo Next.js carga las variables de entorno

Next.js tiene un orden específico para cargar los archivos .env:

1. **.env.local**: Este archivo es el primero en cargarse y siempre se carga, **excepto en producción cuando ya se han definido variables de entorno en el sistema**. Es ideal para variables sensibles que no quieres subir al control de versiones.
2. **.env.[NODE\_ENV]**: Se carga según el valor de NODE\_ENV. Por ejemplo, si NODE\_ENV es production, se cargará .env.production.
3. **.env.[NODE\_ENV].local**: Sobrescribe las variables de .env.[NODE\_ENV] si existen. Por ejemplo, .env.production.local sobrescribe .env.production.
4. **.env**: Este archivo se carga siempre si no se han encontrado las variables en los archivos anteriores.

**Importante:** Las variables de entorno que ya estén definidas en tu sistema (por ejemplo, en el servidor de despliegue) **siempre tendrán prioridad** sobre las definidas en los archivos .env.

## Problemas comunes y soluciones

Dado lo que me comentas, aquí están las razones más probables y cómo abordarlas:

### 1. Prefijo NEXT\_PUBLIC\_ para el lado del cliente

Si intentas acceder a las variables de entorno en el código que se ejecuta en el **lado del cliente** (navegador), estas variables **deben** estar prefijadas con NEXT\_PUBLIC\_. Las variables sin este prefijo solo son accesibles en el lado del servidor (en getServerSideProps, getStaticProps, API Routes, o Middleware).

**Solución:** Asegúrate de que todas las variables que necesitas en el cliente (componentes React, etc.) empiecen con NEXT\_PUBLIC\_.

Ejemplo en .env.production:

NEXT\_PUBLIC\_API\_URL=https://api.tudominio.com

Luego, en tu código:

const apiUrl = process.env.NEXT\_PUBLIC\_API\_URL;

### 2. Variables de entorno en el build (tiempo de compilación) vs. runtime (tiempo de ejecución)

Este es el punto más crítico en producción. Las variables de entorno que se acceden en funciones como getStaticProps o en el next.config.ts se "hornean" en el código durante el proceso de **build** (cuando ejecutas next build). Si estas variables cambian después del build, el Next.js server no las recogerá automáticamente.

**Solución:**

* **Variables que se necesitan en el build (ej. para getStaticProps o next.config.ts):** Deben estar disponibles en el entorno donde se ejecuta next build. Esto significa que si estás desplegando en un servicio como Vercel, Netlify, o un servidor propio, debes configurar estas variables en la plataforma de despliegue antes de que se ejecute el comando next build.
* **Variables que solo se necesitan en el runtime (ej. en getServerSideProps o API Routes):** Estas variables pueden definirse en el entorno de ejecución del servidor.

### 3. Reiniciar el servidor y reconstruir el proyecto

Cada vez que modificas los archivos .env o cambias el entorno (por ejemplo, de development a production), necesitas:

1. **Parar y reiniciar el servidor de Next.js** (npm run dev o npm start).
2. Para cambios que afecten el lado del cliente o se "horneen" en el build, necesitas **reconstruir tu aplicación** (npm run build) y luego **iniciarla** (npm run start).

### 4. Ubicación de los archivos .env

Asegúrate de que tus archivos .env.production y .env.local estén en la **raíz de tu proyecto Next.js**, al mismo nivel que package.json y next.config.ts.

### 5. next.config.ts y las variables de entorno

Por lo general, no necesitas modificar next.config.ts para que Next.js cargue los archivos .env. Next.js lo hace automáticamente. Sin embargo, si necesitas exponer variables de entorno específicas que no tienen el prefijo NEXT\_PUBLIC\_ al lado del cliente (lo cual no es una práctica común ni recomendada para información sensible), o si quieres definir variables directamente en la configuración, podrías usar la propiedad env en next.config.ts.

Ejemplo (no siempre necesario, pero útil para casos muy específicos):

TypeScript

// next.config.ts  
/\*\* @type {import('next').NextConfig} \*/  
const nextConfig = {  
 env: {  
 MY\_CUSTOM\_VARIABLE: process.env.MY\_CUSTOM\_VARIABLE || 'default\_value',  
 },  
};  
  
module.exports = nextConfig;

En este caso, process.env.MY\_CUSTOM\_VARIABLE se haría accesible en el cliente como process.env.MY\_CUSTOM\_VARIABLE. Pero, de nuevo, si es una variable que necesita el cliente, es mejor usar NEXT\_PUBLIC\_.

### 6. Configuración de cross-env

El uso de cross-env es correcto para asegurar que NODE\_ENV se establezca correctamente en diferentes sistemas operativos. Si tu script start ya es cross-env NODE\_ENV=production next start -p ${PORT:-3000}, esto debería estar bien.

### Pasos para depurar:

1. **Verifica NODE\_ENV en producción:** Dentro de una API Route (que siempre se ejecuta en el servidor), puedes hacer un console.log(process.env.NODE\_ENV) para confirmar que realmente es production.
2. **Loguea las variables:** Para depurar qué variables se están cargando, puedes hacer console.log(process.env.YOUR\_VARIABLE\_NAME) en el servidor (por ejemplo, en getServerSideProps o una API Route) y para las variables NEXT\_PUBLIC\_ en un componente de React. **Recuerda eliminar estos console.log antes de ir a producción real.**
3. **Comprueba tu plataforma de despliegue:** Si estás desplegando en Vercel, Netlify, o un servidor propio, asegúrate de que las variables de entorno estén configuradas correctamente en la interfaz o configuración de tu plataforma. Las variables definidas allí a menudo sobrescriben las de los archivos .env en el build.

### Resumen de la solución recomendada:

1. Asegúrate de que las variables que necesitas en el **cliente** tengan el prefijo NEXT\_PUBLIC\_ en tu .env.production.
2. Asegúrate de que las variables que solo necesitas en el **servidor** no tengan el prefijo NEXT\_PUBLIC\_ y estén definidas en tu .env.production o, preferiblemente, directamente en el entorno de tu servidor de despliegue.
3. Después de cualquier cambio en los archivos .env, ejecuta **siempre** npm run build y luego npm run start (o el comando equivalente de tu plataforma de despliegue).
4. Verifica la **ubicación** de tus archivos .env (deben estar en la raíz del proyecto).

Con estos puntos, deberías poder solucionar el problema de carga de variables de entorno en tu aplicación Next.js en producción. ¡Espero que esto te ayude!