Pregunta:

Si deseo que esa conexión abierta no sea total, sino sólo a localhost y a una ip específica, ¿cómo debería configurar bind-address?

Respuesta:

Me alegro de que busques una configuración más específica para mejorar la seguridad. Sin embargo, hay un punto **crítico** que debes conocer sobre bind-address en MySQL, especialmente en versiones antiguas como la 5.0:

**MySQL 5.0 (y versiones anteriores a 8.0.13) solo permite una única dirección IP en bind-address.**

Esto significa que no puedes especificar localhost Y otra IP específica directamente en la línea bind-address. Si lo haces, solo se utilizará la última dirección que indiques.

### ¿Cómo lograr tu objetivo (localhost y una IP específica)?

Dado que no puedes especificar múltiples IPs en bind-address en MySQL 5.0, la estrategia para lograr la conexión desde localhost (dentro del contenedor, para otras aplicaciones del contenedor que necesiten localhost) y una IP específica (la de tu servidor NestJS dentro de la red Docker) es la siguiente:

1. Configura bind-address = 0.0.0.0 (escucha en todas las interfaces).  
   Esta es la solución más práctica y segura cuando necesitas acceso desde varias IPs en un entorno Docker Compose con MySQL 5.0. No te preocupes, el control de acceso real se hará a nivel del usuario de MySQL.  
   En tu my.cnf:  
   [mysqld]  
   bind-address = 0.0.0.0  
     
   ¿Por qué 0.0.0.0 es seguro en este contexto?  
   Aunque 0.0.0.0 significa "escuchar en todas las interfaces", la verdadera seguridad en MySQL no se basa únicamente en bind-address. Se basa en:
   * **Firewall:** Si el contenedor MySQL no está directamente expuesto a internet (solo el puerto 3306 expuesto a localhost en tu máquina host, y las conexiones internas de Docker son controladas por Docker), ya tienes una capa de seguridad. En tu docker-compose.yml al usar ports: - "3306:3306", solo se expone al host, no directamente a la red externa.
   * **Permisos de usuario de MySQL (GRANT):** Esta es la capa de seguridad **más importante** y donde realmente restringes quién puede conectarse.
2. Ajusta los permisos de usuario de MySQL para permitir la conexión solo desde las IPs deseadas.  
   Aquí es donde defines qué IPs están autorizadas a conectarse.
   * **Para localhost (dentro del contenedor MySQL):** Este usuario ya suele estar configurado por defecto para localhost.  
     SQL  
     GRANT ALL PRIVILEGES ON tu\_base\_de\_datos.\* TO 'tu\_usuario'@'localhost' IDENTIFIED BY 'tu\_password';  
     FLUSH PRIVILEGES;  
       
     (Aunque localhost dentro de un contenedor es la IP 127.0.0.1 de ese contenedor, otras aplicaciones que corran en el mismo contenedor o servicios internos de MySQL pueden usarlo).
   * Para la IP específica de tu servidor NestJS:  
     En un entorno Docker Compose, tu servidor NestJS no se conecta desde la IP de tu máquina host, sino desde una IP interna asignada por la red de Docker. La forma más robusta y recomendada es usar el nombre del servicio de MySQL (mysql en tu docker-compose.yml) en la configuración de NestJS, y luego conceder permisos a MySQL usando el wildcard % para el host de ese usuario.  
     ¿Por qué % en el GRANT si quiero una IP específica?  
     Porque la IP interna que Docker asigna al contenedor NestJS puede variar (aunque suele ser estable dentro de la misma red de compose). Al usar % junto con bind-address = 0.0.0.0, le dices a MySQL: "escucha en todas partes, pero para este usuario, la conexión es válida desde cualquier origen".  
     La seguridad se mantiene porque:
     + Tu contenedor MySQL no está expuesto directamente a internet.
     + Solo los contenedores dentro de tu red Docker Compose pueden acceder al nombre de host mysql.
     + Tu aplicación NestJS es la única que tiene las credenciales para ese usuario.

SQL  
-- Crea o modifica el usuario para que tu servidor NestJS pueda conectarse  
-- 'nest\_user' será el usuario que tu aplicación NestJS usará  
-- 'tu\_base\_de\_datos' es la base de datos a la que quieres que acceda  
-- '%' significa "desde cualquier host". Esto es seguro en una red Docker Compose interna.  
GRANT ALL PRIVILEGES ON tu\_base\_de\_datos.\* TO 'nest\_user'@'%' IDENTIFIED BY 'nest\_password';  
FLUSH PRIVILEGES;  
**Nota:** Si realmente tuvieras un escenario donde conoces una IP **estática** y **constante** específica de un contenedor Docker (lo cual no suele ser el caso por defecto en Docker Compose, ya que las IPs son dinámicas aunque estables dentro de la red), entonces podrías usar esa IP específica en el GRANT:SQL  
-- EJEMPLO: Si la IP de tu contenedor NestJS fuera 172.18.0.3 (hipotético y no recomendado para Docker Compose)  
GRANT ALL PRIVILEGES ON tu\_base\_de\_datos.\* TO 'nest\_user'@'172.18.0.3' IDENTIFIED BY 'nest\_password';  
FLUSH PRIVILEGES;  
Pero, como te digo, en Docker Compose es mucho más robusto usar el nombre del servicio (mysql como host en NestJS) y conceder permiso % al usuario en MySQL, ya que el **aislamiento de red de Docker Compose ya se encarga de la seguridad a nivel de red**.

### Resumen de la configuración recomendada para tu caso (MySQL 5.0 en Docker Compose):

1. **En my.cnf (montado como volumen):**  
   [mysqld]  
   bind-address = 0.0.0.0
2. **En MySQL (vía un cliente SQL o un script de inicialización):**  
   SQL  
   -- Usuario para NestJS (reemplaza con tus datos)  
   CREATE USER 'tu\_usuario\_nest'@'%' IDENTIFIED BY 'tu\_password\_nest';  
   GRANT ALL PRIVILEGES ON tu\_base\_de\_datos.\* TO 'tu\_usuario\_nest'@'%';  
     
   -- Si tienes un usuario root o admin para tareas internas desde el mismo contenedor MySQL  
   -- (Este usuario ya suele existir con permisos en 'localhost' o '127.0.0.1')  
   -- GRANT ALL PRIVILEGES ON \*.\* TO 'root'@'localhost' IDENTIFIED BY 'tu\_root\_password';  
     
   FLUSH PRIVILEGES;
3. **En la configuración de NestJS:**  
   TypeScript  
   // En tu configuración de NestJS  
   HOST\_MYSQL: 'mysql', // <-- Nombre del servicio en docker-compose.yml  
   PORT\_MYSQL: 3306,  
   USER\_MYSQL: 'tu\_usuario\_nest',  
   PASSWORD\_MYSQL: 'tu\_password\_nest',  
   DATABASE\_MYSQL: 'tu\_base\_de\_datos',

Esta configuración te proporciona la conectividad necesaria dentro de la red Docker Compose y una seguridad adecuada, ya que las restricciones de acceso se gestionan tanto a nivel de red (por Docker) como a nivel de usuario (por MySQL).