2025-10-24

Monday, April 18, 2022 12:29 PM

Tema: Creación y despliegue de una aplicación NestJS utilizando contenedores Docker.

Objetivos:

- Analizar las diferencias fundamentales entre la virtualización tradicional y la tecnología de contenedores para comprender la eficiencia en el uso de recursos.
- Configurar un entorno de desarrollo inicial instalando el framework NestJS y generando una nueva aplicación.
- Construir una imagen de Docker personalizada para una aplicación NestJS y ejecutarla dentro de un contenedor.
- Administrar el ciclo de vida de los contenedores mediante Docker Desktop, gestionando puertos y entornos simulados (desarrollo, pruebas y producción).

Desarrollo:

Fundamentos: Virtualización vs. Contenedores

Se realizó un análisis comparativo de las arquitecturas de despliegue. Por un lado, la virtualización tradicional depende de un Hipervisor para emular hardware físico, lo que obliga a desplegar un Sistema Operativo completo (Guest OS) en cada máquina virtual. Este modelo genera una carga considerable ("overhead") en el consumo de recursos críticos como memoria RAM, ciclos de CPU y almacenamiento.

En contraposición, se estudió la tecnología de contenedores implementada por Docker, la cual opera mediante un Container Engine. A diferencia de las máquinas virtuales, los contenedores comparten el mismo núcleo (Kernel) del sistema operativo anfitrión, eliminando la necesidad de cargar múltiples sistemas operativos invitados redundantes. Esta arquitectura ligera no solo optimiza el rendimiento, sino que garantiza la consistencia del entorno de desarrollo, asegurando que el software se ejecute de manera idéntica en las estaciones de trabajo de todo el equipo.

Paso 1: Apertura y preparación de la consola

En esta instancia inicial, se procede a abrir la terminal de comandos (CMD) dentro del entorno de desarrollo.

Paso 2: Inicialización del servidor de desarrollo

Tras la generación del proyecto, se procede a acceder al directorio contenedor de la aplicación mediante el comando de navegación (cd). Una vez dentro de la carpeta, se ejecuta el script de arranque del entorno (npm run start). El sistema responde con el mensaje de bienvenida oficial del framework ("Thanks for installing Nest"), confirmando que la instalación de dependencias ha sido exitosa y que el aplicativo base está listo para operar.

Paso 3: Verificación del despliegue en el navegador

Una vez iniciada la ejecución en la terminal, se procede a validar la funcionalidad del servidor web. Para ello, se accede a la dirección de loopback local (localhost) a través del puerto configurado por defecto (3000). El navegador renderiza correctamente la respuesta del servidor mostrando el mensaje "Hello World!", lo cual confirma que el microservicio base se encuentra activo y respondiendo adecuadamente a las peticiones HTTP.

Paso 4: Compilación y preparación del artefacto de despliegue

Previo a la creación de la imagen Docker, se debe asegurar que el proyecto esté optimizado para un entorno de producción. Para ello, se ejecuta el comando npm run build, encargado de transpilar todo el código fuente de TypeScript a JavaScript estándar. Como resultado de este proceso, se genera automáticamente el directorio dist/, el cual debe ser inspeccionado para confirmar la existencia del archivo main.js, punto de entrada que utilizará el contenedor para ejecutar la aplicación.

Paso 5: Definición de la imagen mediante Dockerfile

Se procede a la creación del archivo Dockerfile en la raíz del proyecto. Este documento funciona como un manifiesto de instrucciones que dicta al motor de Docker el procedimiento exacto para empaquetar y construir la imagen de la aplicación.

A continuación se detalla la configuración implementada y la lógica de sus directivas:

1. Definición de la imagen base (versión ligera Alpine)

   FROM node:22-alpine

2. Establecimiento del directorio de trabajo en el contenedor
   WORKDIR /app
3. Transferencia de archivos de configuración de paquetes
   COPY package*.json ./
4. Instalación optimizada de dependencias
   RUN npm install --production
5. Carga del código fuente completo al contenedor
   COPY . .
6. Ejecución del script de construcción (Transpilación)
   RUN npm run build
7. Declaración del puerto de escucha del servicio
   EXPOSE 3000
8. Comando de arranque apuntando al artefacto compilado
   CMD ["node", "dist/main"]

Paso 6: Construcción de la imagen del contenedor
Con el archivo de configuración listo, se procede a generar la imagen Docker desde la terminal. Estando ubicados en la raíz del proyecto, se ejecuta el comando docker build -t mi-app-nestjs .. En esta instrucción, el modificador -t se utiliza para etiquetar (tag) la imagen con el nombre identificativo "mi-app-nestjs", mientras que el punto final (.) es crucial, ya que indica al motor de Docker que el contexto de construcción (y el Dockerfile) se encuentra en el directorio actual.

Paso 7: Ejecución y mapeo de puertos del contenedor
Con la imagen correctamente construida, se procede a iniciar una instancia del contenedor utilizando el comando docker run -p 3000:3000 mi-app-nestjs. En esta instrucción, el parámetro -p 3000:3000 es fundamental, ya que configura el enlace de red entre el entorno virtual y el equipo anfitrión. Esto significa que el puerto 3000 interno del contenedor se mapea directamente al puerto 3000 de la máquina local, permitiendo que la aplicación sea accesible desde el navegador externo.

Paso 8: Personalización de instancias y gestión del ciclo de vida
Para mejorar la administración de los despliegues, es posible asignar identificadores personalizados a las instancias en tiempo de ejecución mediante la bandera --name. A continuación, se presenta un ejemplo donde se despliega un contenedor nombrado explícitamente y se mapea a un puerto alternativo para evitar conflictos:

docker run --name mi-contenedor -p 3001:3000 mi-app-nestjs

Análisis de la instrucción:
- --name mi-contenedor: Asigna el identificador único "mi-contenedor", facilitando su referencia posterior sin depender de IDs aleatorios.
- -p 3001:3000: Redirige el tráfico del puerto 3001 del host al puerto 3000 del contenedor, permitiendo ejecutar múltiples instancias de la misma imagen en paralelo.
- mi-app-nestjs: Especifica la imagen base creada anteriormente.

Esta nomenclatura simplifica significativamente la ejecución de tareas administrativas de mantenimiento, permitiendo utilizar comandos directos como:

- docker stop mi-contenedor para detener la ejecución.
- docker start mi-contenedor para reiniciar el servicio.
- docker logs mi-contenedor para inspeccionar la salida de la consola y depurar errores.

Nota importante: Es fundamental monitorear la ocupación de puertos en la máquina local. Si un puerto requerido ya está en uso por una instancia anterior irrelevante, se debe proceder a detener y eliminar dicho contenedor para liberar el recurso antes de lanzar uno nuevo.

Conclusiones:
- **Optimización de recursos mediante contenerización:** La arquitectura de Docker demostró ser significativamente más eficiente que la virtualización tradicional. Al compartir el núcleo del sistema operativo anfitrión y prescindir de un sistema operativo invitado completo (Guest OS), los contenedores reducen drásticamente la carga sobre el hardware

(CPU y RAM), permitiendo un despliegue más ligero y rápido de los servicios.

- **Estandarización y portabilidad del software:** La implementación del archivo Dockerfile permitió encapsular la aplicación NestJS y sus dependencias en una unidad inmutable. Esto garantiza la consistencia absoluta del entorno, eliminando el problema clásico de "funciona en mi máquina"; el software se ejecuta con el mismo comportamiento exacto tanto en la estación de desarrollo local como en los servidores de producción.
- **Gestión simplificada del ciclo de vida:** El uso de herramientas de orquestación como Docker Desktop facilitó la administración de los contenedores, permitiendo gestionar de manera intuitiva el mapeo de puertos y la segregación de ambientes (Desarrollo, Pruebas y Producción). Esto reduce la complejidad de configuración de redes y evita conflictos de dependencias entre distintos proyectos que coexisten en el mismo equipo.

Bibliografía:

- Docker, "Descripción general de Docker," *Documentación de Docker*. [En línea]. Disponible: https://docs.docker.com/get-started/overview/
- Red Hat, "¿Qué es la contenerización?," *Temas de Red Hat*. [En línea]. Disponible: https://www.redhat.com/es/topics/cloud-native-apps/what-is-containerization
- NestJS, "Primeros pasos - Introducción," *Documentación de NestJS*. [En línea]. Disponible: https://docs.nestjs.com/