Dockerfile para Backend Node.js

Configuración de Contenedor para Aplicación IoT

Propósito General

Este Dockerfile define la configuración para containerizar la aplicación backend desarrollada en Node.js, que incluye servicios de Express, WebSocket y MQTT. Sigue las mejores prácticas de seguridad y optimización para entornos de producción.

Estructura del Dockerfile

Imagen Base

Utiliza node:20-alpine como imagen base, proporcionando:

Versión LTS: Node.js 20 con soporte a largo plazo

Distribución Alpine: Imagen minimalista que reduce significativamente el tamaño final

Seguridad: Menor superficie de ataque al incluir solo paquetes esenciales

Metadatos del Contenedor

Incluye labels descriptivos para:

Mantenimiento: Información de contacto del equipo

Descripción: Propósito específico de la aplicación

Versión: Control de versiones del contenedor

Gestión de Usuarios y Seguridad

Usuario No-Root

Implementa principio de mínimo privilegio mediante:

dockerfile

```
RUN addgroup -g 1001 -S nodejs && \
  adduser -S nodejs -u 1001
```

Crea grupo y usuario dedicado con IDs específicos

Previene ejecución con privilegios de root

Mitiga riesgos de seguridad en caso de compromiso

Cambio de Propietario

dockerfile

RUN chown -R nodejs:nodejs /app
USER nodejs

Asegura permisos correctos en el directorio de trabajo

Cambia contexto de ejecución al usuario no-privilegiado

Optimización de Capas Docker

Estrategia de Cache

dockerfile

COPY backend/package*.json ./
RUN npm ci --only=production
COPY backend/src ./src

Copia independiente: Separa archivos de dependencias del código fuente

Cache eficiente: Reutiliza capas cuando no cambian las dependencias

Instalación optimizada: npm ci para instalación reproducible y más rápida

Limpieza de Cache

dockerfile

RUN npm cache clean --force

Reduce tamaño de imagen final

Elimina archivos temporales innecesarios

Configuración de la Aplicación

Directorio de Trabajo

dockerfile

WORKDIR /app

Establece directorio consistente dentro del contenedor

Facilita operaciones de archivo relativas

Variables de Entorno

dockerfile

ENV NODE_ENV=production
ENV PORT=3000

NODE_ENV: Configura entorno de producción para optimizaciones

PORT: Define puerto por defecto, configurable via docker-compose

Exposición de Puertos

dockerfile

EXPOSE 3000

Documenta puerto que la aplicación utiliza

Compatible con configuración de docker-compose

Health Check

dockerfile

HEALTHCHECK --interval=30s --timeout=3s --start-period=5s --retries=3 \
  CMD node -e "require('http').get('http://localhost:3000/health', (res) => {
process.exit(res.statusCode === 200 ? 0 : 1) })"

Parámetros de Configuración

Intervalo: Verificación cada 30 segundos

Timeout: 3 segundos máximo por verificación

Periodo inicial: 5 segundos de gracia al iniciar

Reintentos: 3 fallos consecutivos marcan el contenedor como unhealthy

Lógica de Verificación

Realiza solicitud HTTP al endpoint /health

Verifica código de estado 200 (OK)

Utiliza comando inline de Node.js para minimizar dependencias

Comando de Inicio

dockerfile

CMD ["node", "src/server.js"]

Ejecuta el punto de entrada principal de la aplicación

Utiliza formato array para evitar shell interpolation

Mantiene proceso principal en primer plano

Características de Seguridad

Best Practices Implementadas

Usuario no-root: Ejecución con privilegios mínimos

Imagen minimalista: Alpine reduce superficie de ataque

Dependencias de producción: Solo paquetes necesarios

Health checks: Monitoreo automático de salud

Cache cleaning: Eliminación de datos sensibles

Consideraciones de Producción

Secret management: Las credenciales deben inyectarse via secrets o variables de entorno

Logging: Configuración adecuada para agregación de logs

Monitoring: Integración con sistemas de monitoreo externos

Resource limits: Límites de CPU/memoria definidos en docker-compose

Flujo de Construcción

Base layer: Imagen Alpine con Node.js 20

Dependencies layer: Instalación de paquetes npm

Source code layer: Código de la aplicación

Configuration layer: Permisos y configuración

Runtime layer: Health check y comando de inicio

Compatibilidad con Docker Compose

El Dockerfile está diseñado para integrarse con la configuración de docker-compose:

Puerto 3000 alineado con la configuración del servicio

Variables de entorno sobreescribibles

Health checks para orquestación automática

Esta configuración proporciona una base sólida, segura y optimizada para desplegar la aplicación backend en cualquier entorno containerizado, desde desarrollo hasta producción.