**Dockerfile: как контейнеризировать Node.js приложение**

**📘 2.1 Что такое Dockerfile?**

**Dockerfile** — это текстовый файл со списком инструкций, которые говорят Docker:

* какую систему использовать
* как установить зависимости
* как скопировать код
* какие порты открыть
* какую команду запускать

**🛠️ 2.2 Пример Node.js проекта**

У тебя есть такой проект:

pgsql

КопироватьРедактировать

docker-node-app/

├── Dockerfile

├── package.json

├── index.js

**📄 package.json**

json

КопироватьРедактировать

{

"name": "docker-node",

"version": "1.0.0",

"scripts": {

"start": "node index.js"

},

"dependencies": {

"express": "^4.18.2"

}

}

**📄 index.js**

js

КопироватьРедактировать

const express = require('express');

const app = express();

app.get('/', (req, res) => res.send('Привет из Docker!'));

app.listen(3000, () => console.log('Сервер работает на порту 3000'));

**📄 2.3 Dockerfile — ПОЛНЫЙ РАЗБОР ПОСТРОЧНО**

Dockerfile

КопироватьРедактировать

# 1. Используем базовый образ Node.js

FROM node:18

🔹 FROM — это то, с чего всё начинается.  
Здесь мы используем официальный образ Node.js версии 18.  
Docker скачает его из Docker Hub.

Dockerfile

КопироватьРедактировать

# 2. Устанавливаем рабочую директорию внутри контейнера

WORKDIR /app

🔹 Все команды будут выполняться в каталоге /app внутри контейнера.  
Это как cd /app внутри Linux-системы.

Dockerfile

КопироватьРедактировать

# 3. Копируем package.json и package-lock.json

COPY package\*.json ./

🔹 COPY — копирует файлы из твоего компьютера в контейнер.  
Мы копируем только зависимости **до установки npm** — это ускоряет кеширование Docker.

Dockerfile

КопироватьРедактировать

# 4. Устанавливаем зависимости

RUN npm install

🔹 RUN выполняет команду внутри контейнера.  
Здесь мы устанавливаем зависимости из package.json.

Dockerfile

КопироватьРедактировать

# 5. Копируем оставшиеся файлы

COPY . .

🔹 Всё остальное (включая index.js, .env, и т.д.) копируется в рабочую директорию.

Dockerfile

КопироватьРедактировать

# 6. Открываем порт 3000

EXPOSE 3000

🔹 Это указание, что приложение **слушает порт 3000**, и его нужно пробросить наружу.  
Важно: EXPOSE сам по себе не открывает порт. Это скорее документация.

Dockerfile

КопироватьРедактировать

# 7. Стартовая команда

CMD ["node", "index.js"]

🔹 Это **команда по умолчанию**, которая запустится при старте контейнера.  
Пишется как массив — это формат exec (надёжнее, чем строка).

**⚡ 2.4 Альтернатива: использование npm start**

Dockerfile

КопироватьРедактировать

CMD ["npm", "start"]

Если у тебя в package.json указан start-скрипт, это будет более универсально.

**🧼 2.5 Добавим .dockerignore**

Чтобы не тянуть лишние файлы (node\_modules, логи):

📄 .dockerignore

lua

КопироватьРедактировать

node\_modules

npm-debug.log

.env

**🏗️ 2.6 Сборка и запуск контейнера**

**🛠️ Сборка:**

bash

КопироватьРедактировать

docker build -t my-node-app .

**🚀 Запуск:**

bash

КопироватьРедактировать

docker run -p 3000:3000 my-node-app

Теперь приложение доступно по адресу:  
👉 <http://localhost:3000>

**🏋️ 2.7 Оптимизация Dockerfile (Alpine)**

**📦 Использование лёгкого образа:**

Dockerfile

КопироватьРедактировать

FROM node:18-alpine

**Alpine** — это суперлёгкий Linux (~5MB), подходит для продакшна.  
Однако, иногда не хватает нужных библиотек (например, libc6).

**🧪 2.8 Проверка контейнера**

🔍 Проверить, что контейнер запущен:

bash

КопироватьРедактировать

docker ps

📜 Посмотреть логи:

bash

КопироватьРедактировать

docker logs <container\_id>

🛑 Остановить:

bash

КопироватьРедактировать

docker stop <container\_id>

🗑 Удалить:

bash

КопироватьРедактировать

docker rm <container\_id>

**📦 2.9 Пример итогового Dockerfile**

Dockerfile

КопироватьРедактировать

FROM node:18-alpine

WORKDIR /app

COPY package\*.json ./

RUN npm install

COPY . .

EXPOSE 3000

CMD ["node", "index.js"]

**📌 2.10 Вывод главы**

Теперь ты знаешь:

* Как работает Dockerfile
* Как собрать и запустить контейнер
* Как оптимизировать образ
* Как запускать Node.js приложение в контейнере