**Оптимизация и безопасность Node.js + Docker в продакшне**

В этой главе ты научишься:

* Использовать **multistage Dockerfile**
* Подключать **HTTPS и Nginx**
* Настраивать **Production Mode**
* Уменьшать размер образов
* Повышать **безопасность деплоя**

**🧱 9.1 Multistage Dockerfile — зачем?**

В обычном Dockerfile:

* Сначала ставятся dev-зависимости (например, для тестов, ESLint и т.д.)
* Потом копируется всё
* Это увеличивает размер финального образа

**Multistage** позволяет:  
✅ Использовать одну стадию для сборки  
✅ Вторую — для финального образа с **только нужными файлами**

**📦 9.2 Пример Multistage Dockerfile**

Dockerfile

КопироватьРедактировать

# ====== Этап 1: build ======

FROM node:18 as build

WORKDIR /app

COPY package\*.json ./

RUN npm install

COPY . .

# Здесь можно запустить тесты, линтер, билд

# RUN npm run build

# ====== Этап 2: production ======

FROM node:18-alpine

WORKDIR /app

COPY --from=build /app ./

ENV NODE\_ENV=production

RUN npm prune --production

EXPOSE 3000

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

**⚖️ 9.3 Сравнение размеров**

| **Dockerfile** | **Размер** |
| --- | --- |
| Обычный | ~350MB |
| Multistage + Alpine | ~80MB |

💡 Экономия — огромная, особенно при CI/CD и масштабировании

**🔐 9.4 Безопасность контейнера**

✅ **Минимизируй доступы**  
По умолчанию ты root внутри контейнера — это плохо. Добавь:

Dockerfile

КопироватьРедактировать

RUN addgroup app && adduser -S -G app app

USER app

✅ **Не сохраняй секреты в контейнере**  
Используй .env или secrets manager (Vault, AWS Secrets)

✅ **Ограничь доступ к контейнеру**  
В docker-compose:

yaml

КопироватьРедактировать

network\_mode: bridge

✅ **Только нужные порты**

Dockerfile

КопироватьРедактировать

EXPOSE 3000

**🌐 9.5 Подключение HTTPS через Nginx**

Docker сам по себе **не умеет в HTTPS** — он отдаёт HTTP.  
Для защищённого трафика — ставим **Nginx в отдельный контейнер**.

**⚙️ 9.6 Пример docker-compose с Nginx**

cpp

КопироватьРедактировать

production-stack/

├── docker-compose.yml

├── nginx/

│ ├── nginx.conf

│ └── ssl/

│ ├── cert.pem

│ └── key.pem

├── app/

├── Dockerfile

├── index.js ...

**📄 nginx.conf**

nginx

КопироватьРедактировать

server {

listen 443 ssl;

server\_name yourdomain.com;

ssl\_certificate /etc/nginx/ssl/cert.pem;

ssl\_certificate\_key /etc/nginx/ssl/key.pem;

location / {

proxy\_pass http://app:3000;

proxy\_http\_version 1.1;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

}

}

**📦 9.7 docker-compose.yml**

yaml

КопироватьРедактировать

version: '3.8'

services:

app:

build: ./app

expose:

- "3000"

networks:

- web

nginx:

image: nginx:latest

volumes:

- ./nginx/nginx.conf:/etc/nginx/conf.d/default.conf

- ./nginx/ssl:/etc/nginx/ssl

ports:

- "443:443"

depends\_on:

- app

networks:

- web

networks:

web:

**🔐 9.8 HTTPS с Let's Encrypt (автоматически)**

👉 Используй **Caddy** или **Traefik**:

* Автоматическая генерация и обновление SSL
* Простая настройка, например:

yaml

КопироватьРедактировать

image: caddy

ports:

- "80:80"

- "443:443"

volumes:

- ./Caddyfile:/etc/caddy/Caddyfile

**🔧 9.9 Дополнительные оптимизации**

| **Практика** | **Польза** |
| --- | --- |
| NODE\_ENV=production | Убирает dev-зависимости |
| npm prune --production | Уменьшает node\_modules |
| COPY --chown=app:app | Безопасность |
| HEALTHCHECK | Контроль жизни сервиса |
| restart: unless-stopped | Перезапуск при сбое |
| resource limits | mem\_limit: 512m — ограничение ресурсов |

**🔐 9.10 Пример: продакшн-Compose**

yaml

КопироватьРедактировать

services:

app:

build: .

environment:

- NODE\_ENV=production

restart: unless-stopped

healthcheck:

test: curl -f http://localhost:3000 || exit 1

interval: 10s

retries: 3

nginx:

image: nginx

ports:

- "443:443"

...

**✅ Результат**

Теперь твой проект:

* Оптимизирован
* Надёжно контейнеризован
* С безопасным HTTPS
* С минимальным размером и зависимостями
* Готов к CI/CD и продакшену