МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра Вычислительной техники

ОТЧЕТ

по лабораторной работе № 2
по дисциплине «Web программирование»
Тема: «Установка и настройка среды разработки и исполнения Web-приложения»

Студент гр. 3311 Баймухамедов Р. Р. _____

Преподаватель Калмыков М.А. _____

Санкт-Петербург
2025

Цель работы

Знакомство с основами сборки и развёртывания web-проекта в Docker на стеке Nginx + Node.js (Express): инициализация проекта npm, контейнеризация backend-сервиса и статического фронта, настройка reverse-proxy и проверка работы приложения.

Выполнение лабораторной работы

Инициализируем бэкенд проекта, выполнив `npm init -y`, добавив зависимость express, создав server.js с эндпоинтом GET /api/hello (возвращающий json)

```
PS C:\GitHub\course-3\web\lab_01\app> npm init -y
Wrote to C:\GitHub\course-3\web\lab_01\app\package.json:

{
  "name": "app",
  "version": "1.0.0",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node server.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": ""
}
```

Код server.js

```
const express = require("express");
const app = express();

app.get("/api/hello", (req, res) => {
  res.json({ message: "Hello nginx from node.js api" });
});

const PORT = 3000;
app.listen(PORT, () => console.log(`Node app listening on ${PORT}`));
```

Код Dockerfile

```
FROM node:alpine

RUN apk add --no-cache \
      bash curl wget unzip zip tar git \
      ca-certificates openssl tzdata jq dumb-init \
 && update-ca-certificates

WORKDIR /app
```

```
COPY package*.json ./

RUN if [ -f package-lock.json ]; then \
      npm ci --omit=dev ; \
    else \
      npm install --omit=dev express ; \
    fi

COPY server.js ./

EXPOSE 3000
ENTRYPOINT ["dumb-init","--"]
CMD ["node","server.js"]
```

Подготовим фронтэнд часть

В каталоге src/ создадим index.html, который запрашиваем /api/hello и отображает ответ

Код index.html

```
<!doctype html>
<meta charset="utf-8">
<title>Lab 1 (Nginx + Node)</title>
<h1>Hello, world 👋</h1>
<p>It’s 1st lab of web prog. Served by <b>Nginx</b>.</p>
<p id="api">Loading API…</p>
<script>
  (async () => {
    const el = document.getElementById('api');
    try {
      const r = await fetch('/api/hello', { headers: { Accept: 'application/json'
} });
      if (!r.ok) throw new Error(`HTTP ${r.status}`);
      const ct = r.headers.get('content-type') || '';
     const data = ct.includes('application/json') ? await r.json() : await r.text();
       el.textContent = (data && typeof data === 'object' && data.message) ?
data.message
                      : (typeof data === 'string' ? data : JSON.stringify(data));
    } catch (e) {
      el.textContent = 'API error: ' + e.message;
    }
  })();
</script>
```

Изменим конфигурацию nginx

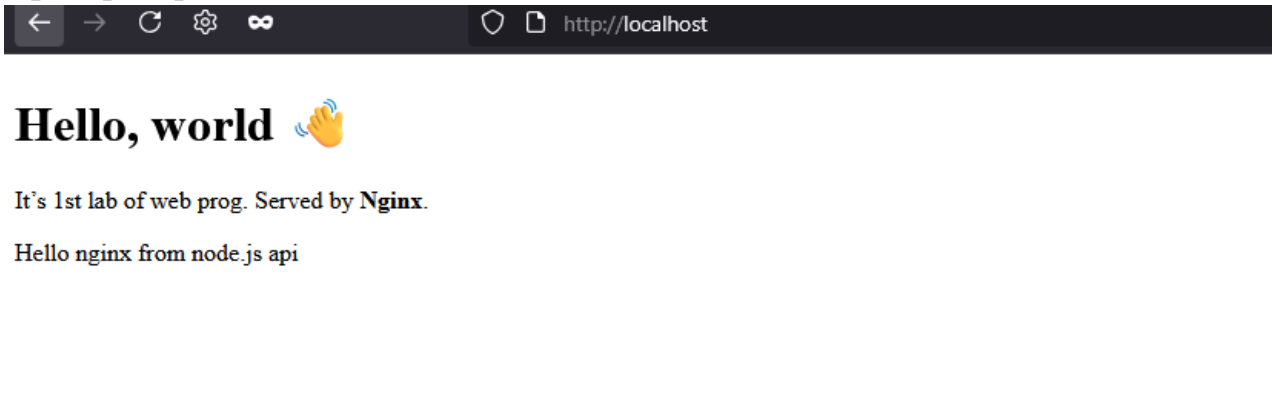
Статика будет раздаваться из /usr/share/nginx/html. Проксирование location /api/ -> http://app:3000

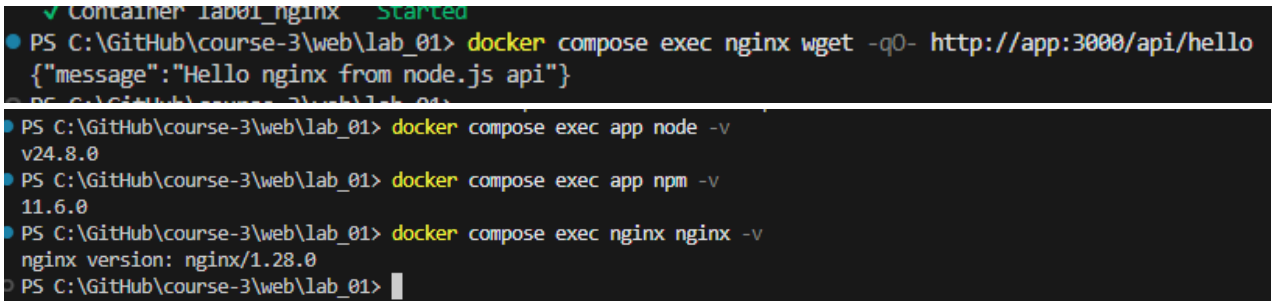
Следующим шагом настроим контейнеризацию сервисов

В docker-compose.yml описаны два сервиса: app (Express) и nginx. Их запуск происходит с помощью команды `docker compose up -d --build`
Код docker-compose.yml

```yaml
services:
  app:
      build: ./app
      container_name: lab01_node
      expose:
        - "3000"
      environment:
        - NODE_ENV=production
  nginx:
    image: nginx:stable-alpine
    container_name: lab01_nginx
    ports:
      - "80:80"
      - "8080:80"
    volumes:
      - ./nginx/default.conf:/etc/nginx/conf.d/default.conf
      - ./src:/usr/share/nginx/html
    depends_on:
    - app
```

Проверим работоспособность





Заключение

В ходе работы создан и собран минимальный web-проект на Node.js (Express) с фронтом на статических файлах, раздаваемых через Nginx, и настроенным reverse-proxy для API. Проект упакован и запущен в Docker Compose, что обеспечило воспроизводимое окружение и простое управление сервисами. Получены практические навыки: работа с npm/Express, базовая конфигурация

Nginx, организация маршрутизации / и /api/, а также контейнеризация как современная альтернатива сборке WAR и деплою на Tomcat.